PROGRAMMERINGSARBETE PÅ INTEL 3000

HANS HANSSON
INGVAR LARSSON

RE-164 oktober 1975
Inst. för Reglerteknik
Lunds Tekniska Högskola

PROGRAMERINGSARBETE PÅ INTEL-3000

INNEHÅLL

		sid
INLEDNING	abstract	1
HÅRDVARAN	allmänt INTEL-3000	2
	realisering	7
	avbrottshantering	11
INSTRUKTIONSLISTOR	KMV version 1	16
	KMV version 2	22
ASSEMBLER	allmänt	25
	PDP-11:s Assembly Instr.	25
	definiering av Macro	26
	positionsräknaren	27
	Macro Defnitioner	27
	införande av nya instr.	31
	kommando till assemblern	31
	objektremsan	32
MONITOR		39
SAMMANFATTNING		41

APPENDIX

Flödesschema för Monitorrutiner	A1
Monitoranvända minnesceller i SP	A12
Listning av Monitor	A13
Listning av Avbrottsrutin	A30
Operationstider	A31
Demonstrationsprogram	A32

INLEDNING

Detta examensarbete utfördes under våren och sommaren 1975 i samarbete med instutionen för Reglerteknik vid LTH och Kockums Mekaniska Verkstad i Malmö.
Handledare var Gustaf Olsson och Leif Andersson från LTH samt Bengt Jönsson från Kockums (Loadmasteravd.)
Arbetet ingick som en del i ett projekt att digitalt övervaka lastning och lossning av tankfartyg. För detta ändamål används ett mikrodatorsystem av typ Intel 3000.
Iinledningsskedet var våra uppgifter tämligen löst definierade, så småningom utkristaliserades emellertid följande fyra delar:

1/ Orientering i Intel 3000:s uppbyggnad samt den aktuella hårdvarurealiseringen.

2/ Modifieringsarbete av befintlig instruktionslista.

3/ Göra en cross-assembler mellan PDP-11 och Intel 3000.

4/ Skriva en bildskärmshanterad monitor med remsloader.

ABSTRACT

Största delen av examensarbetet har bestått av att ta fram hjälpmedel för programutveckling. Bristen på sådana hjälpmedel är det största problemet med microdatorer, oberoende av hårdvarurealiseringen.
Den cross-assembler som framtagits, med en PDP 11 som värddator och KMV-version 2 som måldator, kommer att vara till stor hjälp när man skriver program eftersom man med en macroassembler höjer språknivån avsevärt.
Monitorn användes vid inläsning av den objektremsa som erhålls vid assembleringen, samt för att testa ut program.
Ett optimalt utnyttjande av hårdvaran i INTEL 3000 resulterar i en instruktionsuppsättning liknande den som framtagits för KMV-version 2. Det vore därför önskvärt att kretstillverkaren (INTEL) tog fram en "standardinstruktionslista". Denna kan sedan användaren utvidga med nästan obegränsat kraftfulla instruktioner genom egen microprogrammering.

The main part of this work consists of the developing of software support. The lack of such support is the main problem with micro computers, independent of the hardware realization.
The cross assembler developed, having a PDP 11 as host computer and KMV-version 2 as target computer, will be a great help when writing assembler programs, while the macro assembler raises the language to a higher level.
The monitor is used to load the memory with the binary object code from the paper tapes as they are received from the assembler. The monitor is also used to test the program.
An optimal use of the INTEL 3000 hardware results in an instruction list rather similar to the one developed for KMV-version 2. Due to this fact it would be a great help to all users if the manufacturer (INTEL) would supply a "standard instruction list". It would then be easy for the user to expand the list with powerful special instructions for his own purpose.

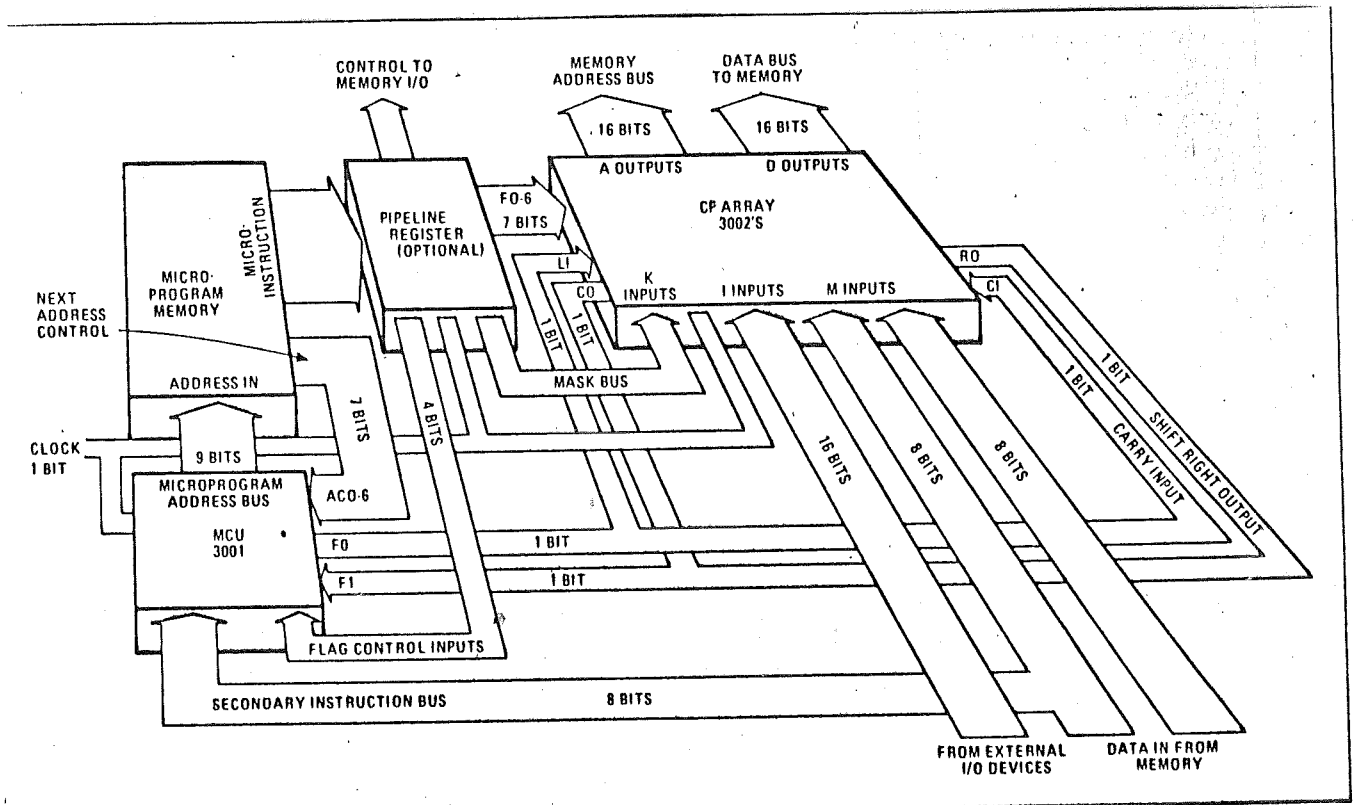
HÅRDVARAN

Intel-3000

Intel-3000 är en familj av bipolära LSI-kretsar, som förenklar konstruktionen av mikroprogramerade CPU-er. Familjen består av :

3001 Microprogram Control Unit (MCU)
3002 Central Processing Element (CPE)
3003 Look-Ahead Carry Generator (LCG)
3212 Multi-Mode Latch Buffer
3214 Priority Interupt Control Unit (ICU)
3216 Noninverting Bidirectional Bus Driver
3226 Inverting Bidirectional Bus Driver
3301 Schottky Bipolar ROM (256x4)
3304 Schottky Bipolar ROM (512x8)
3601 Schottky Bipolar PROM (256x4)
3604 Schottky Bipolar PROM (512x8)

Dessa kan organiseras enligt nedanstående figur.

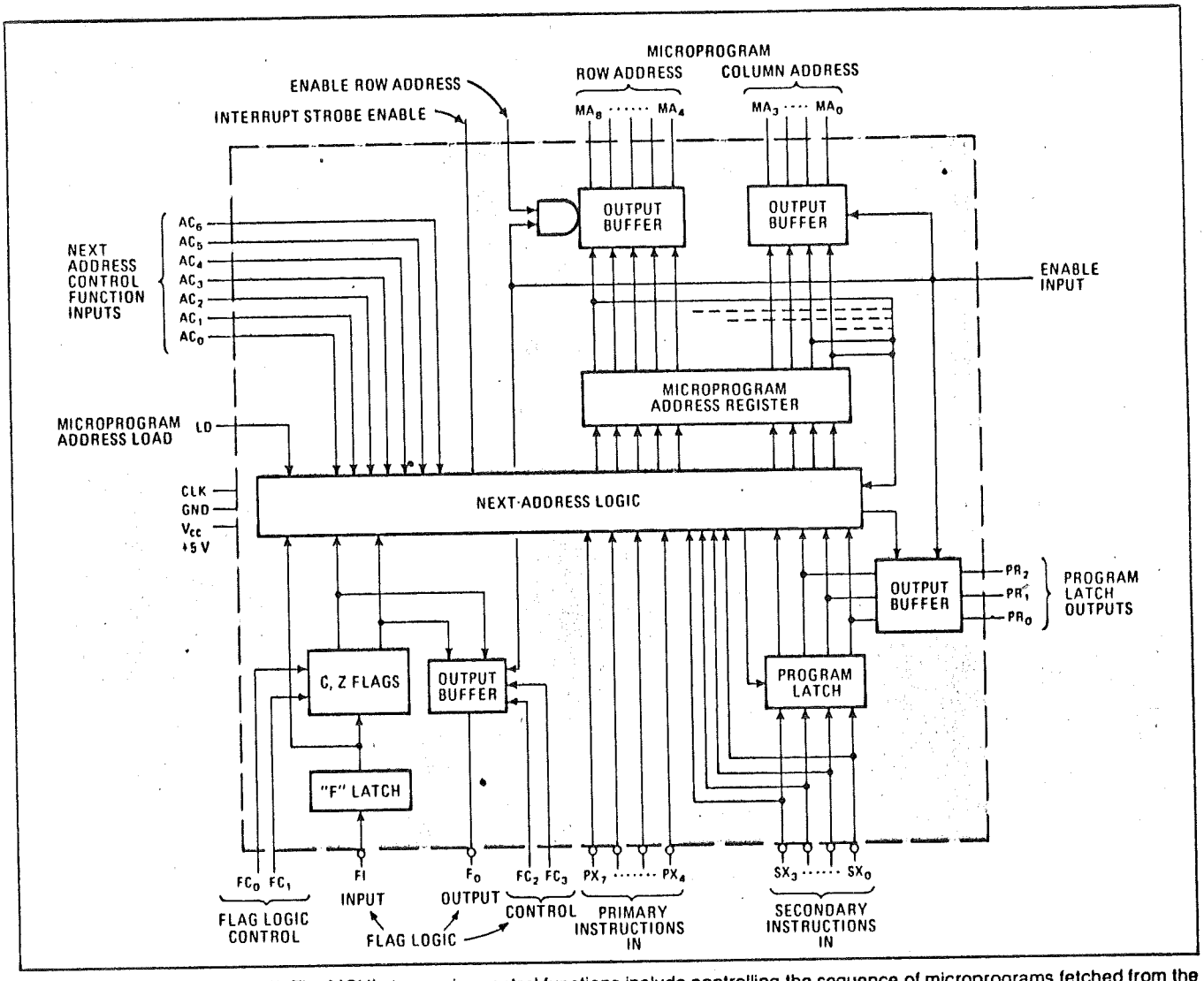


1. Bipolar microcomputer. Block diagram shows how to implement a typical 16-bit controller-processor with new family of bipolar computer elements. An array of eight central processing elements (CPEs) is governed by a microprogram control unit (MCU) through a separate read-only memory that carries the microinstructions for the various processing elements. This ROM may be a fast, off-the-shelf unit.

Kort presentation av några familjemedlemmar i Intel-3000 familjen

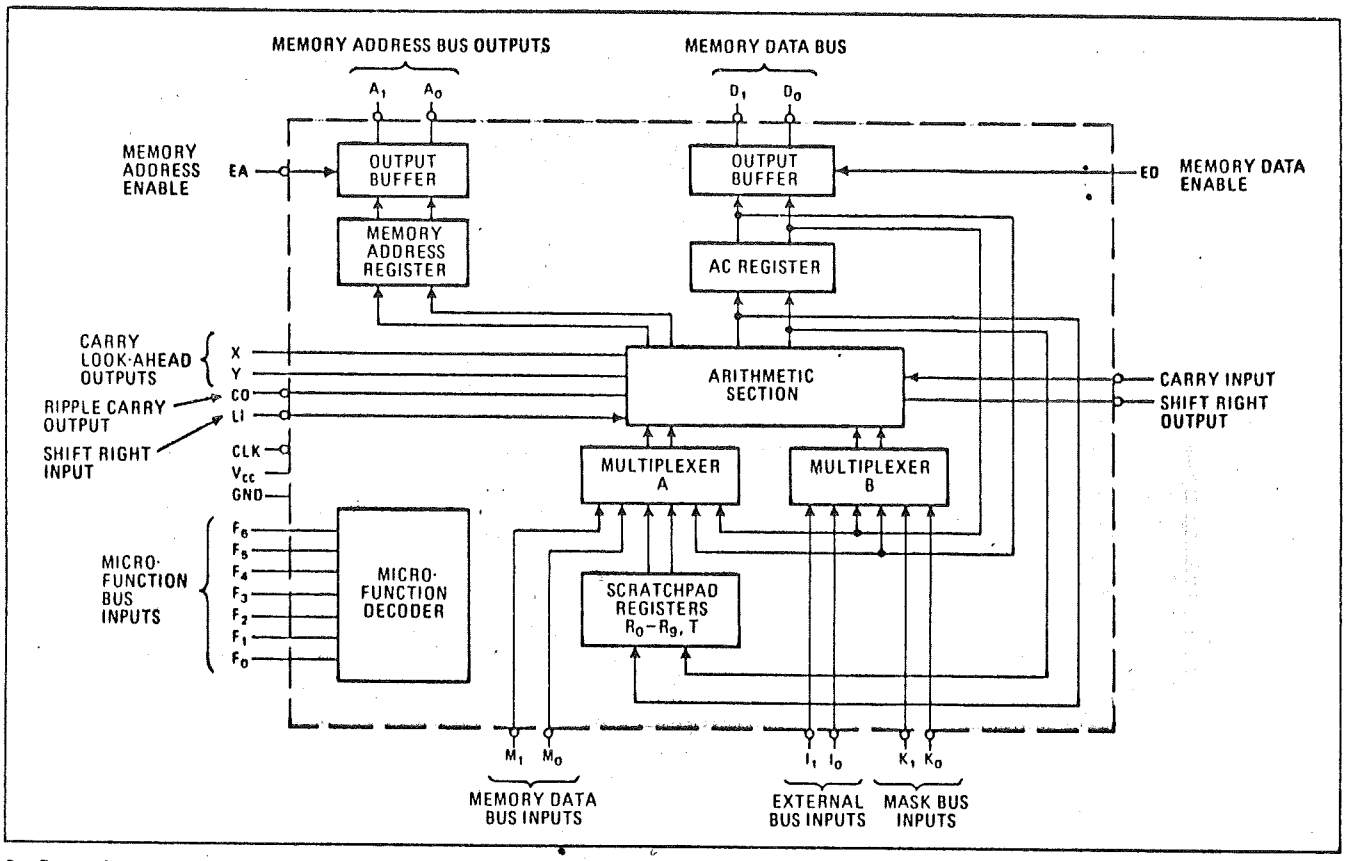
3001 MCU Mikroprogramkontrollenheten kontrolerar ordningen som mikroinstuktionerna hämtas från mikroprogramminnet.

MCU:n är uppbyggd enligt nedanstående figur.



6. Microprogram control unit. The MCU's two major control functions include controlling the sequence of microprograms fetched from the microprogram memory, and keeping track of the carry inputs and outputs of the CP array by means of the flag logic control.

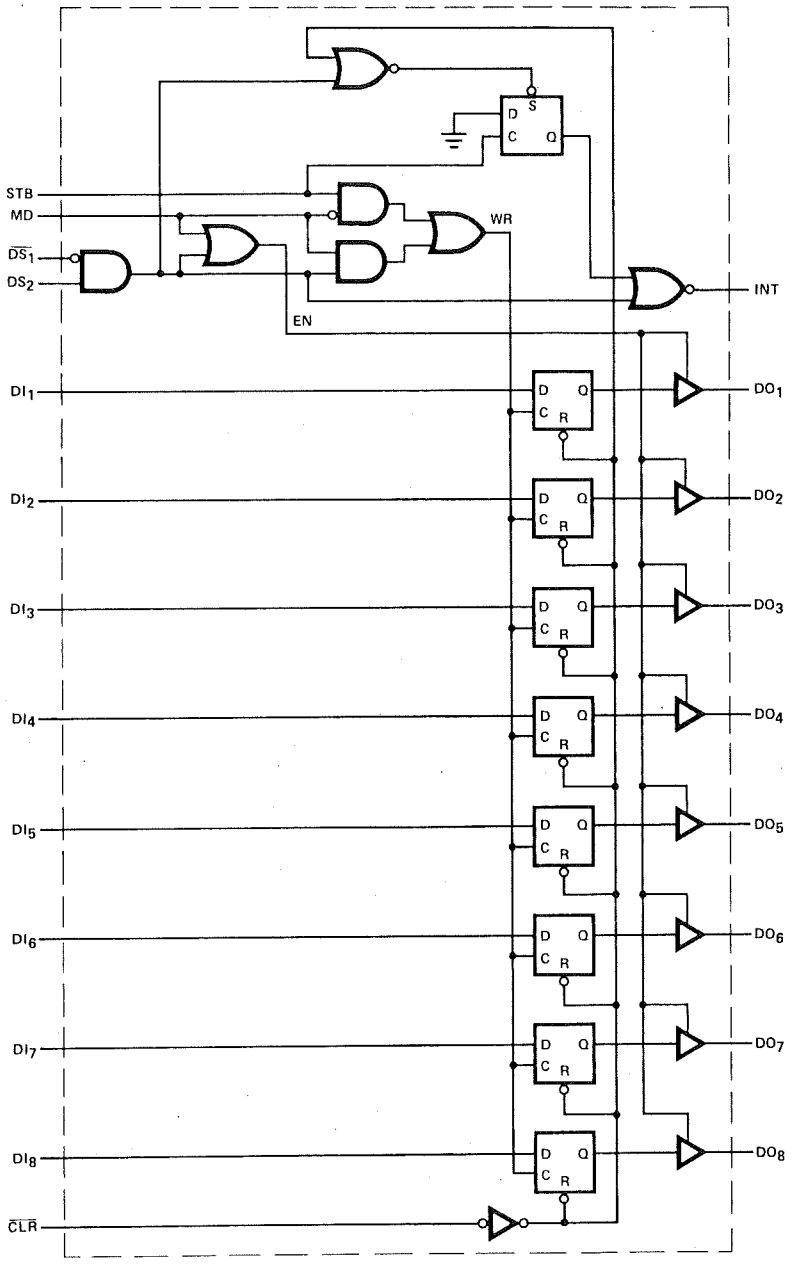
3002 CPE Processorenheten innehåller alla kretsar som behövs för en 2-bitars processor. Man kan direkt sammankoppla flera CPE:er till valfri ordlängd. CPE:n har tre databussar för inmatning,en adressbus och en databus för utmatning. Den innehåller förutom aritmetisk-logisk enhet tio generella register.
CPE:n är uppbyggd enligt nedanstående figur. (nästa sida)



3. Central processing element. This element contains all the circuits representing a two-bit-wide slice through a small computer's central processor. To build a processor of word width N, all that's necessary is to connect an array of N/2 CPEs together.

3003 LCG Look-Ahead Carry Generator används för att snabba upp de aritmetiska operationerna. Vid logiska operationer används LCG:n till att generera ett ordvis inklusivt eller.

3212 MULTI-MODE LATCH BUFFER är en åtta bitars latch med "three-state" utgångsbuffert och inbyggd urvalslogik.



3212 Logic Diagram

3214 ICU ger avbrottskapacitet i flera nivåer för processorer byggda med 3000-serien. ICU:n får en asynkron avbrottspuls från MCU:n eller en bit i microprogramminnet och genererar en synkron avbrottspuls och en avbrottsvektor, som kan påföras MCU:n eller CPE:erna för att identifiera avbrottskällan. Varje ICU kan ta emot avbrott från åtta enheter samt ge avbrott i åtta olika nivåer.

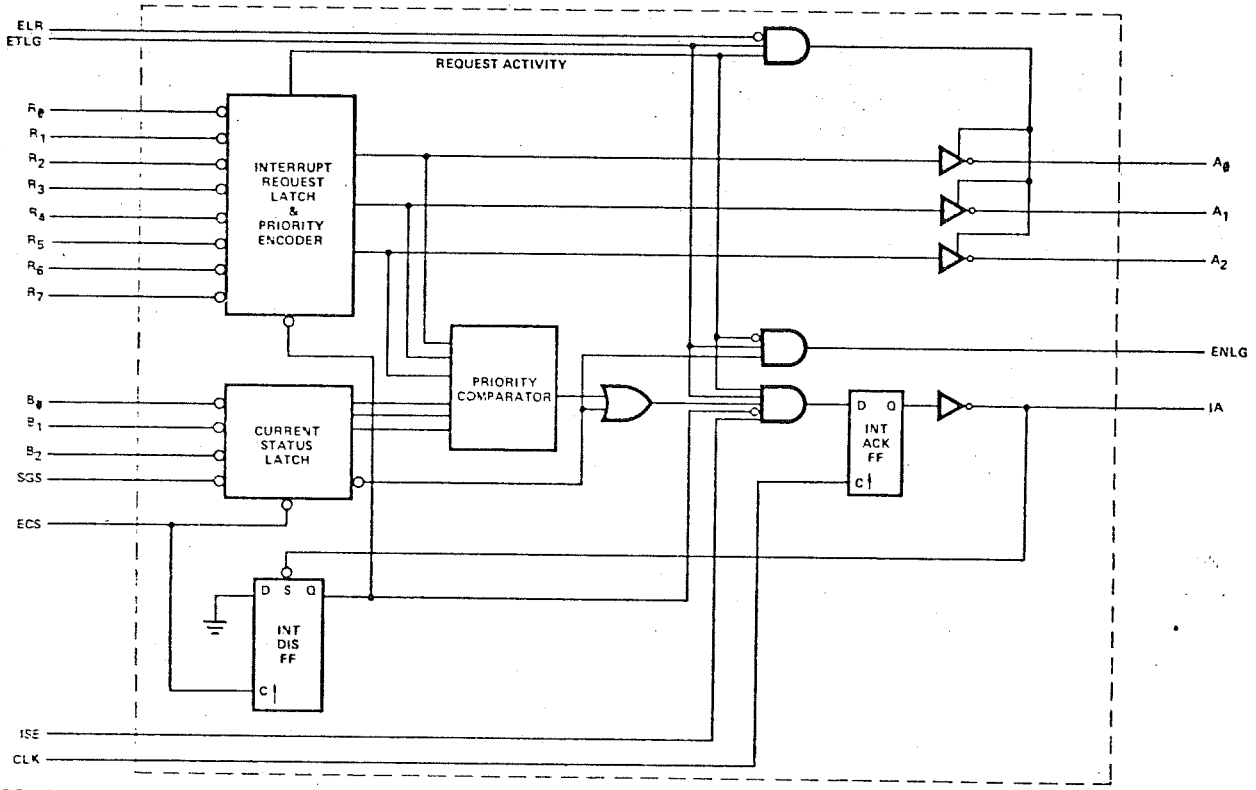
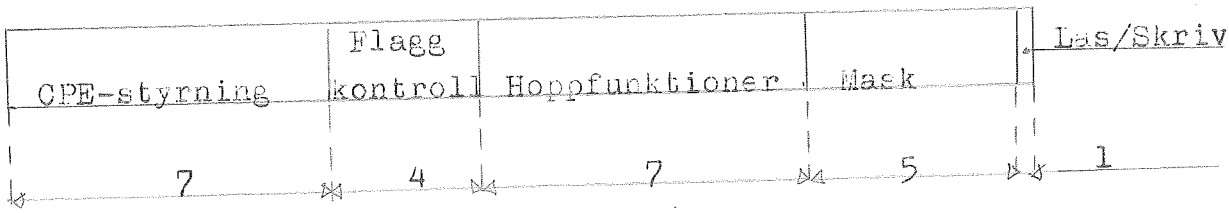


Figure 1. 3214 Block Diagram.

REALISERING

CPU:n är sammansatt av åtta 3002 CPE:er,en 3001 MCU,tre 3212 Latchar,en 3003 LOG,512 ord om 24 bitar bipolärt microprogramminne med en acesstid på 50nS,samt kretsar för in- och ut-matning.Bitarna i microprogramminnet används enligt följande:



Den sista biten anvands också till las/skriv då man vill ha tre alternativ: 1/ Las, 2/ Skriv, 3/ Open Collektor. Maskbitarna går till K-bussen enligt:

K 0	K 1-K 7	K 8-K 11	K 12-K14	K 15

De mest signifikanta åtta bitarna i I-bussen är anslutna parallellt med bitarna i K-bussen,de övriga åtta bitarna är anslutna parallellt med M-bussen,detta för att möjliggöra instruktioner med direkt operand.
Alla bitarna ,utom de som styr hoppfunktionerna,går via D-vippor.Denna "pipelinig" beror på att man då kan få en överlappning så att nasta microinstruktion kan lasas in i D-vipporna medan den nuvarande microinstruktionen utföres.Detta förfarande ger snabbare exekvering,då man inte behöver vänta på microprogramminnet utan kan ta microinstruktionen "direkt" i D-vipporna.
Utav CPE:ernas tio register är fyra åtkomliga för programmeraren.Register R7 används som programräknare och register R6 används som stackpekare.

CPU:n är förseedd med ett Scratch Pad minne om 256 ord. Detta har en acesstid på 40nS.
De två utgångarna PRO-PR1 på MCU:n är anslutna på CPE:ernas F0-F1 för att man i instruktionen skall kunna adressera de olika registerna.

EXEKVERING AV INSTUKTION

På rad 0 column 15 i microprogramminnet börjar microprogrammet för uthämtning av en ny instruktio.Uthämtningen börjar med att register R7 överföres till MAR och adresserar minnet.Efter en klockpuls så finns instruktionen på M-bussen.Med JPR hoppar man till kolumnen där instruktionens microprogram börjar.Om instruktionen innehåller adressmod ,tas adressen fram med ettt microprogram som nås via JLL.Därefter tas operanden in och instruktionen utföres.Microprogrammet för alla instruktionerna avslutas med ett JZR:15,alltså ett hopp till början av microprogrammet för uthämtning av ny instruktion.

MICROPROGRAMMINNETS ORDLÄNGD

Den parameter,som bestämmer hur det mesta av det andra ser ut,är microprogramminnets ordlängd.Med en kort ordlängd får man en dålig instruktionslista och svårigheter att hantera extra hårdvara.Med en lång ordlängd blir instruktionslistan bra och man kan lagga till extra hårdvara,såsom avbrottshantering,utan problem.
MCU:n behöver fyra bitar till flaggkontroll och sju bitar till bestämma nästa adress,alltså totalt elva bitar.Eventuellt kan man behöva flera bitar till MCU:n för att styra sidvsl i microprogramminnet.

CPE:n behöver sju bitar för att kontrollera micro-funktion och val av register.
Det behövs alltså 18 bitar för att kontrollera CPE:erna och MCU:n, vilket är den absolut minsta ordlangd som kan användas.
För att kunna maska tal och sätta räknare behövs bitar som styr K-bussen. De krav, som ställs i en 16 bitars processor är:
1/ Man skall kunna maska av den mest signifikanta biten för att kunna avgöra tecken: K=077777
2/ Man skall kunna maska av den övre eller den undre byten för att exempelvis kunna skilja pa operation och operand: K=00377 eller K=177400.
3/ Man skall kunna sätta räknare till ± 15 exempelvis vid multiplikation: K=000017 eller K=177761.
En rättfram realisering skulle kräva fem bitar enligt:

Bit 1	K0
Bit 2	K1-K7
Bit 3	K8-K11
Bit 4	K12-K14
Bit 5	K15

Detta förfarande används i maskinen vi arbetat med, men är egentligen ett slöseri med bitar. Eftersom det endast är fem olika kombinationer som behövs skulle man med ett kombinatoriskt nät kunna orana detta med hjalp av tre bitar vilket ger åtta olika kombinationer. Detta förfarande drar dock ner snabbheten något vilket är motivet till att det inte används i kockums maskinen.

Som synes av instruktionslistan är endast fyra av de nio registerna åtkomliga för programmeraren.Detta beror på att MCU:n har tester pa fyra eller två bitar i operationskoden,vilket gör det lämpligt att ha fyra adressmoder och fyra register.Man kan emellertid också ha andra kombinationer men då måste man lagga till bitar för att kunna manipulera de multiplexrar som blir nödvändiga och eventuellt också några D-vippor för att kunna hålla kvar informationen om operationskodens övriga bitar förutom de fyra som lagras i MCU:n.Detta förfarande torde vara intressant i 24 och 32 bitars maskiner där ordlängden är så stor att fler än åtta bitar kan användas till operationskod.

Vid avbrottshantering kan man,som i denna maskinen, hantera avbrottsenheten som en yttre enhet,men för att få upp snabbheten och för att kunna tillåta avbrott i flera nivåer är det nödvändigt att manövrera detta via bitar i microprogrammet.Dessutom är det då bäst att använda en hårdvarustack,för att lagra undan avbrottsstatus, vilket också kräver en bit.

Om man bygger en maskin för en speciell uppgift kan man tänka sig att lägga in subrutiner i microprogrammet. Man måste då ha en micro-stack för att lagra återhoppsadressen,denna måste då skötas via två bitar i microprogramminnet.

Slutligen behövs bitar för att ge information om läsning eller inskrivning i minnet.Eventuellt behövs också en bit för att kunna hantera in och utmatning till perifera enheter.

Sammanfattningsvis kan man säga att för små maskiner borde 24 bitars ordlängd vara tillräckligt i microprogramminnet men på större maskiner torde man behöva 28 eller eventuellt 32 bitar.

AVBROTTSHANTERING

När det gäller avbrottshantering kan man tänka sig många olika system,från det enklaste med rästan ingen hårdvara till det mest sofistikerade med mycket hårdvara.

System 1:Ett enkelt system visas på fig. 2:1.Vid ett JZR:15 genereras en strobpuls på ISE,denna tillsammans med en avbrottsbegäran triggar D-vippan,som tvingar MCU:n,genom en puls på ERA,in i ett avbrottsmicroprogram.Dessutom blockeras nya avbrott genom att den asynkrona vippan sätts.I avbrottsmicroprogrammet lagras PC och eventuellt andra register undan varefter ett hopp sker till ett speciellt avbrottsprogram.Där testas först vilken enhet,som gav avbrott varefter en, för varje enhet speciell,avbrottsrutin exekveras.När detta är färdigt sker ett återhopp till det avbrutna programmet och avbrott tillåtes på nytt genom att den asynkrona vippan nollställs,genom en utinstruktion.

System 2: Metoden att mjukvarumässigt låta testa alla avbrytande enheter är långsam.Det ligger därför nära till hands att låta detta skötas av hårdvara.Man förfar då enligt fig. 2:2.I avbrottsmicroprogrammet adresserar man prioritetsavkodaren,som då ger ut ett statusord,vilket är den oktala koden för den enhet som har högst prioritet av de enheter som begärt avbrott.Detta statusord påverkar sedan hoppadressen så att man direkt kommer till avbrottsprogrammet för den aktuella enheten.Med detta systemet får man alltså en viss prioritering i och med att den enhet som har högst prioritet kommer att behandlas först,emellertid kan inte en enhet med högre prioritet avbryta en enhet med lägre prioritet vars avbrottsprogram just exekveras.För att kunna göra detta måste man anvanda

System 3: Detta är ett system där man tillåter avbrott

i flera nivåer.Informationen om nuvarande status lagras i stack,lämpligen en separat hårdvarustack för att slippa alltför komplicerade microprogram och för att spara tid och plats i primärminnet.Om man till exempel har 16 avbrytande enheter behövs endast en fyra bitars stack. Om man har 32 bitars ordlängd skulle det ge dålig utnyttjning att lägga stacken i det vanliga minnet.Dessutom skulle många ord i onödan behöva reserveras till stacken.En fördel med att ha stacken i primärminnet är att man då kan mjukvarumässigt förhindra avbrott genom att ge stackens topp ett lågt värde.
Ett avbrott går först genom prioritetsavkodaren varefter statusordet jämföres med nuvarandestatus,om prioriteten är högre får man ett avbrott,den nya statusen lagras i stacken och avbrottsrutinen för enheten börjar exekveras.Om prioriteten är lägre händer inget.

MINNETS ACESSTID

Primärminnets acesstid är av största betydelse för hur fort instruktionerna exekveras.Genom ett smart utnyttjande av registerna kan emellertid betydelsen minskas. Dessuton kan man i en specialbyggd maskin skriva microprogram,som är så långa att tiden att adressera minnet blir liten i förhållande till tiden det tar att exekvera instruktionen.Som exempel kan nämnas att MPF tar upp till 50µS med 50nS minne och upp till 53µS med 1µS minne,alltså en skillnad på endast 6%.
Om man inför subrutiner i microprogrammet och hårdvarustack kommer betydelsen av minnets acesstid att minska ytterligare.
På den aktuella maskinen är minnets acesstid 50nS, vilket beror på att det är så få ord ,som behövs,att det inte blir mycket dyrare än ett långsamt minne.På en större maskin får man emellertid tänka sig att använda ett långsammare minne av kostnadsskäl.

AVBROTTSHANTERARE

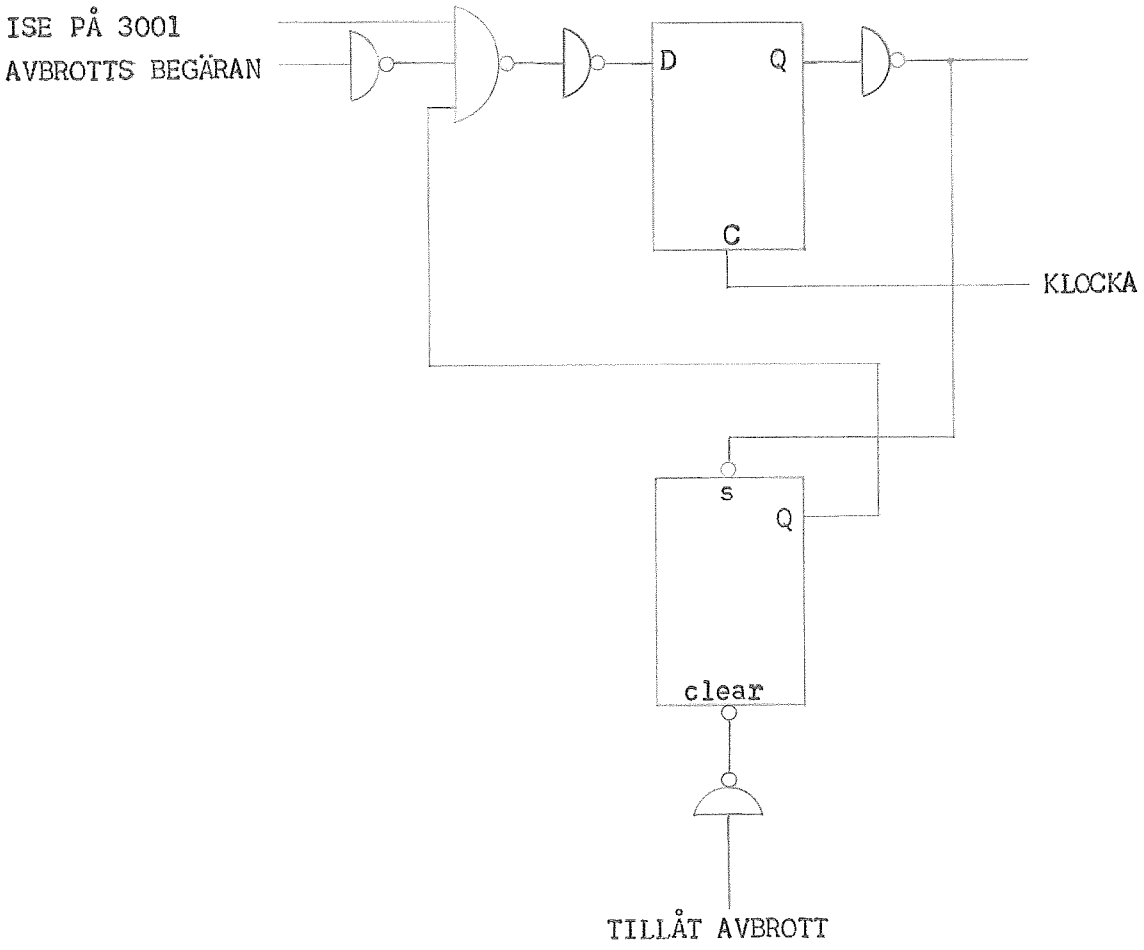


FIG. 2.1

PRIORITETSAVKODARE

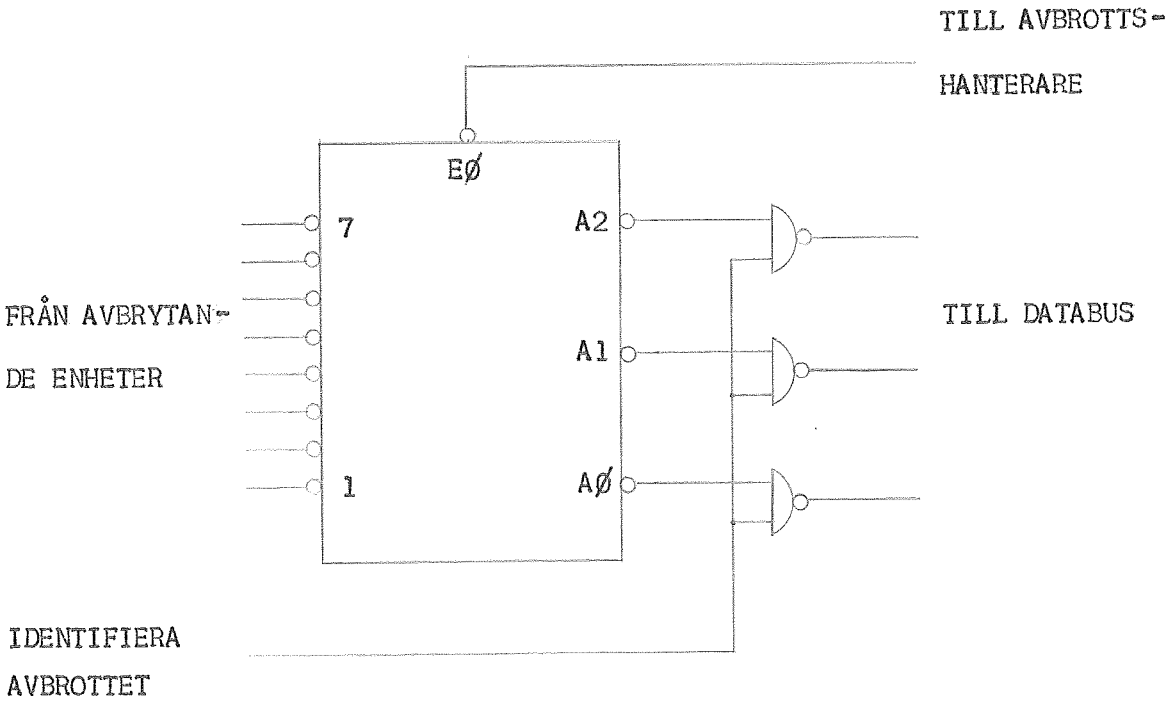
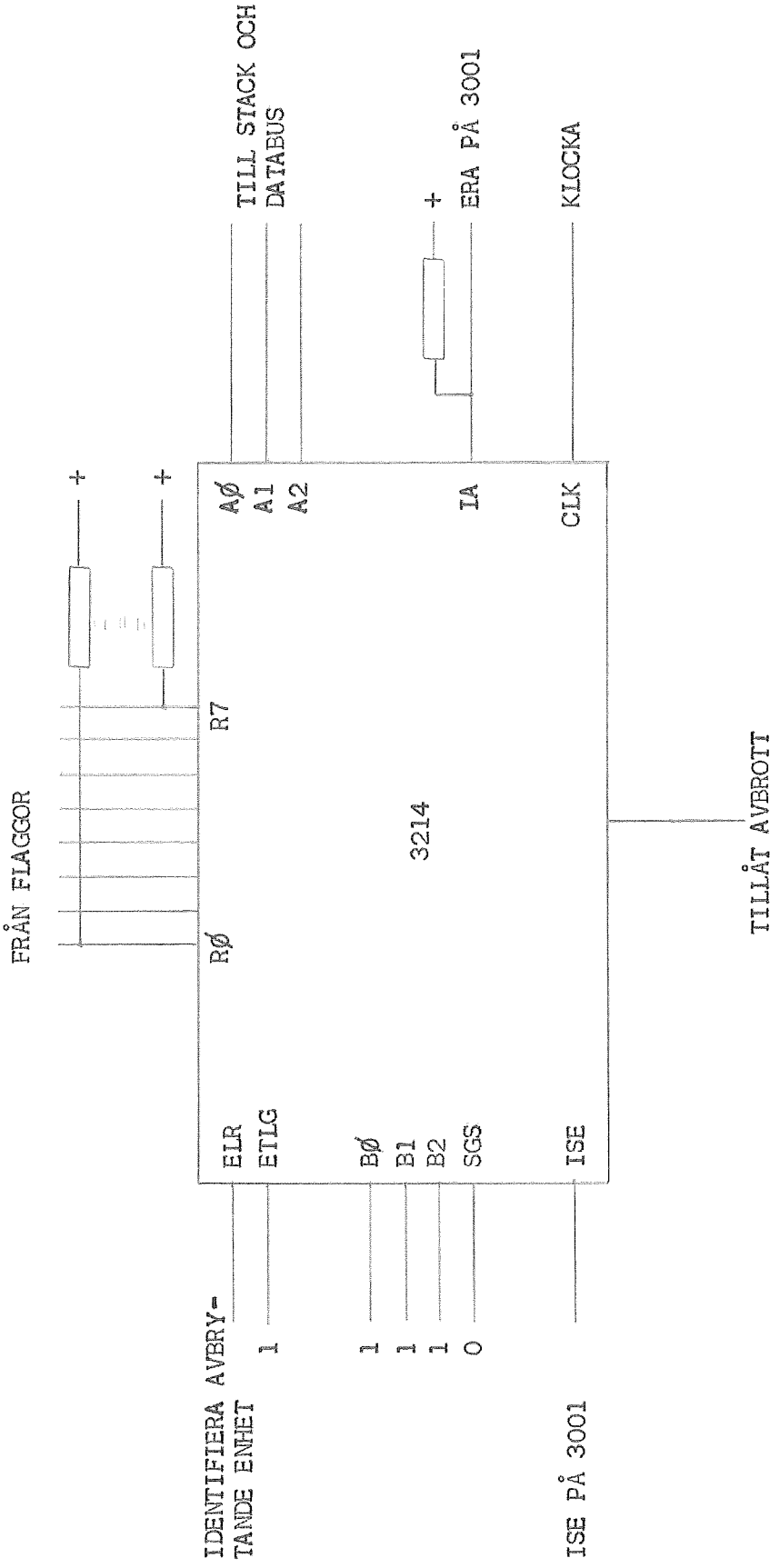


FIG. 2.2


FRÅN FLAGGOR
+
+
R7
RØ
AØ
A1
A2
IA
CLK
TILL STACK OCH DATABUS
+
ERA PÅ 3001
KLOCKA
3214
ELR
ETLG
BØ
B1
B2
SGS
ISE
IDENTIFIERA AVBRYTANDE ENHET
1
1
1
1
0
ISE PÅ 3001
TILLÅT AVBROTT

INSTRUKTIONSLISTOR

Genom att Intel 3000 är en mikroprogramerbar dator kan instruktionslistan skräddarsys för sin speciella tillämpning. I vårt aktuella fall ska datorn huvudsakligen utföra beräkningsprogram av typ tredjegradspolynom. Det är således viktigt att maskinen kan göra snabba flyttalsberäkningar.

Vid examensarbetets början fanns redan en instruktionslista framtagen. Denna benämns KMV version 1, instruktionerna med förklaringar är listade på sidorna 17 - 19.

Instruktionerna använder i regel åtta bitar till operationskod och åtta bitar till adress eller direkt operand.I denna version används tre adresseringsmoder, dessa är: 1/ Basrelativ, 2/Självrelativ, 3/Direkt adress till sida Ø (SP).

De olika adresseringsmoderna används emellertid inte konsekvent till aktuella instruktioner.

CPU:n har åtta register åtkommliga för användaren. Dessa är funktionsmässigt lika, men R7 används som programräknare (PC) samt R6 som basadress vid basrelativ adressernig. Vid flyttalsoperationer upptar den ena operanden RØ och R1, den andra operanden R4 och R5.

Flyttalet är uppdelat på de två orden enligt: 24 bitars mantissa och 8 bitars exponent.

Som framgår av instruktionslistan använder en del instruktioner tre bitar till registerval (åtta register) , medan andra endast använder en eller två bitar till detta. En klass av instruktioner arbetar naturligtvis utan registerval (ex. NOP).

Exekveringstiden för en instruktion erhålls som antalet mopar x klockpulsintrevallet, som är 150 ns. Ex LOAD : 12x150 ns = 1.8 μs.

+/- i instruktionslistans displacementfält betyder att displacement anges med tecken dvs inom talområdet $+177_8$ - -200_8 eller $+127_{10}$ - -128_{10}.

Instruktionslista KMV version 1

antal mopar	mnemonamn	opkod	displ.	register	adress rel.	beskrivning av funktion
12	LOAD	10000RRR	+/- D	0-7	R6	innehållet i cell R6+D till register R_n (R_n ges av RRR)
13	LOADF	10001A1R	+/- 0-255	0,1 4,5	R6,SP	flyttalsladda register RØ,R1 om R=Ø, R4,R5 om R=1 operanden från R6+D om A=Ø, från SP om A=1 (dvs från 0-255)
10	STORE	10010RRR	+/-	0-7	R6	lagra innehållet i R_n till cell R6+D
11	STOF	10011A00	+/- 0-255	0,1	R6,SP	lagra flyttal från RØ och R1 till R6+D och R6+D+1 om A=Ø till SP om A=1
10	LDII	00010RRR	+/-	0-7	R7	ladda register R_n från cell R7+D , dvs.rel.PC (självrelativt)
	JSR	1100CCRR	0-255	0-3	(n+1)	subrutinanrop . anropet utförs om CC=ØØ och R_n=Ø,CC=Ø1 och R_n=negativt, CC=1Ø och R_n=positivt, ovilkorligt om CC=11.Adressen till subrutinen ges i nästa cell (n+1). Återhoppsadressen lagras i SP (o-255)
6	LDI	00110RRR	I	0-7		ladda register R_n direkt med I
8-9	ADDI	00100RRR	I	0-7		addera direkt I till R_n resultat i R_n
8	LRA	10100RRR	00000000	0-7		kopiera $R_Ø$ till R_n
7	LAR	10100RRR	10000000	0-7		kopiera R_n till $R_Ø$
7	RADD	10110RRR	00000000	0-7		registeraddition. R_n+$R_Ø$ till $R_Ø$
8	RSUB	10110RRR	10000000	0-7		registersubtraktion. $R_Ø$- R_n till $R_Ø$
12	ADDM	01100ARR	+/- 0-255	0-3	R6,SP	minnesaddition. innehållet i cell R6+D adderas till R_n om A=Ø innehållet i cell i SP adderas till R_n om A=1, resultat i R_n
13	SUBM	01101ARR	+/- 0-255	0-3	R6,SP	minnessubtraktion. analogt med minnesaddition
11	JMP	1101CCRR	+/-	0-3	R7	hopp enligt samma regler som vid subrutinanrop, hoppet är självrelativt dvs adressen ges av R7+D

antal mopar	mnemonamn	op.kod	displ.	register	adress rel.	beskrivning
7+2n	SHR	111100RR	n shift	0-3		antal steg i 2-komplement högerskift logiskt
7+2n	SHL	111101RR	n shift	0-3		" vänsterskift logiskt
9+2n	ROR	111110RR	n shift	0-3		" rotera höger
9+2n	rol	111111RR	n shift	0-3		" rotera vänster
6	AND	111000RR		0-3		logiskt OCH mellan R_n och $R_\emptyset$ resultat i $R_\emptyset$
6	OR	111001RR		0-3		logiskt ELLER mellan R_n och $R_\emptyset$ resultat i $R_\emptyset$
6	XNOR	111010RR		0-3		exklusivt NOR mellan R_n och $R_\emptyset$ resultat i $R_\emptyset$
9	COMPL	111011RR		0-3		teckenbyte av R_n i tvåkomplement, resultat i R_n
4	HLT	00001000				halt
8	MOVF	01001110		0,1		kopiera flyttal från RØ och R1 till R4 och R5
25 µs	ADDF	01000000		0,1 4,5		flyttalsaddition mellan register RØ,R1 och R4,R5 resultat i RØ,R1
25 µs	SUBF	01000011		0,1 4,5		flyttalssubtraktion analogt med flyttalsaddition
50 µs	MPYF	01000010		0,1 4,5		flyttalsmultiplikation analogt med flyttalsaddition
	IN	00001100	0-127	Ø		inenhet nr(0-127) till register RØ
	OUT	00001101	0-127	Ø		utenhet nr(0-127) laddas från register RØ
	ANI	01001100	I	Ø		logiskt OCH direkt mellan RØ och I resultat i RØ
15 µs	MPY	01001101		0,1		onormerad multiplikation mellan RØ och R1 resultatet till RØ resultatet mindre än 2^{16}
	NORM	0100		Ø		normalisering av RØ till RØ, exponent + bias 128 till R1 (bit 0-7)
	DENORM	0100				motsatsen till normaliseringen ovan

antal mopar	mnemonamn	op.kod	displ.	register	adress rel.	beskrivning
23 µs	DIV			0,1		registerdivision mellan RØ och R1 resultat i RØ
9	LDS	01011RRR	0-255	0-7	SP	ladda register R_n från SP
7	STS	01010RRR	0-255	0-7	SP	lagra innehållet i R_n i SP
8	RET	01000001	0-255			hämtar återhoppsadressen vid subrutin från SP

Förslag till modifieringar av instruktionslistan

Som framgått av instruktionslistan (version 1) är denna inte helt konsekvent och systematiskt uppbyggd. För att förenkla och samtidigt utöka instruktionslistan föreslår vi följande förändringar:

1/ Inför generellt fyra adresseringsmoder:

a/ Direkt sida Ø

b/ Basrelativ adressering

c/ Indirekt adressering

d/ Självrelativ adressering

2/ Använd generellt två bitar för att utpeka vilket register som ska användas.

3/ Flytta basregistert från R6 till R2.

4/ Utöka registerinstruktionerna så att även R_n kan vara resultatregister.

För att förenkla och effektivisera subrutinhopp och avbrott föreslår vi dessutom:

5/ Resevera de första cellerna i minnet för en stack, med stackpekare i cell Ø.

6/ Kraftfull avbrottsinstruktion.

Motiveringar till modifieringsförslag

1/ För att programeringen ska vara enkel bör instruktionslistan vara systematiskt uppbyggd. Om man använder två bitar till adresseringsmod får man fyra möjliga adresseringsmoder. Basrelativ, direkt sida Ø och indirekt adressering kan anses självklara i vår tillämpning. För den fjärde moden står valet mellan självrelativ och autoinkrement. Vi föredrar självrelativ eftersom den gör programmet relokerbart samt förenklar datahämtningen från en subrutin.

2/ I makroinstruktionerna används enbart RØ, R3, R6 och R7. Det är därför onödigt att resevera tre bitar för registerval. Vi föreslår därför att man använder två bitar konsekvent. Detta förenklar också mikroprograme ringen (framgår av hårdvaruavsnittet) samt spar mikroprogramminne.

3/ Två bitar i registerval medför att att basregistret måste vara något av RØ - R3 för att vara laddbart, RØ och R1 används till flyttalsoperationer, välj R2 eller R3. Vi kan då även göra JMP på registret och det uppfyller då vilkoren för att kallas indexregister.

4/ För att effektivt kunna utnyttja registerna bör man även kunna lagra resultat från operationer i detta.

5/ Med JSR (subrutinhopp) som den nu finns får man själv lagra återhoppsadressen i SP. Detta går bra så länge man själv påkallar ett subrutinhopp, men med avbrott gör man ett påtvingat subrutinhopp och då fungerar denna metod inte så bra. Det vore därför lämpligt att ha en stack, där återhoppsadressen lagras.

6/ Eftersom maskinen inte ska förses med DMA, måste det finnas kraftfulla avbrottsinstruktioner, så att inte komunikation med snabba yttre enheter tar för lång tid. De bör se ut enligt nedan.

A Avbrottsbegäran

a/blockera nya avbrott
b/ lagra PC i stack
c/ lagra register i stack (RØ - R3)
d/ identifiering av avbrottsenhet
e/ hopp till,för avbrottsenheten, speciell servicerutin.

B Återhopp

a/ återställ register enligt b/- c/ ovan
b/ möjliggör avbrott

KMV version 2

Efter en tids testande och funderande framtogs en andra version av instruktionslista till Intel 3000. Denna presenteras på följande sidor.

Flertalet instruktioner är liknande de i version 1, i version 2 används emellertid två bitar generellt till adresserindsmod samt två bitar till registerval. Adresseringsmoderna blev:

a/ Direkt adress till sida Ø, A = ØØ, benämns i assembly med S
b/Självrelativ adressering, A = Ø1, " P
c/ Basrelativ adressering, A = 1Ø, " R
d/ Indirekt adressering, A = 11 " I

Vid direkt adress till sida Ø tolkas displacementfältet som ett positivt tal (Ø - 255), som direkt ger adressen.

Vid självrelativ adressering tolkas displacementfältet som ett tal med tecken ($+127_{10}$ - 128_{10}) som adderas till PC och ger adressen.

Vid basrelativ adressering används R3 i stället för PC, i övrigt som självrelativ .

Vid indirekt adressering tolkas displacementfältet som ett positivt tal, som pekar ut den cell vari slutadressen finns.

Indirekt adressering gör även autoinkrement på innehållet av cellen som ger adressen, efter det att adressen har beräknats.

Maskinen kommer i sin slutgiltiga utforming att arbeta med minne till stor del av typ PROM, härvid fungerar naturligtvis indirekt adressering som vanlig indirekt, dvs utan inkrementering.

Vidare har det tillkommit en stack som ligger i SP med början i cell Ø. Stackpekare är register R6. I stacken lagras automatiskt återhoppsadressen vid subrutiner.

Den föreslagna kraftfulla avbrottsinstruktionen föreslogs i ett skede då det var meningen att ett skivminne skulle tillkomma. Detta var i ett senare skede inte längre aktuellt, varför även instruktionen utgick.

I instruktionslistan anges inom parantes mnemonamnet som används i assemblyn.

Eventuellt kommer instruktionslistan att kompleteras med ytterligare instruktioner.

KMV version 2

mnemonamn	op.kod	displ.	beskrivning
LOAD (LD)	0001AARR	D	ladda register R_n från minnescell
STORE (ST)	0010AARR	D	lagrar ut innehållet från R_n till minnescell
LOADF (LDF)^X	0100AA00	D	flyttalsladda register RØ och R1 från celler
STOREF (STF)^X	0100AA01	D	flyttalslagra från RØ och R1 till celler
ADDM (ADM)	0101AARR	D	addera innehållet från cell till R_n resultat i R_n
SUBM (SUM)	0111AARR	D	subtrahera analogt med ADDM ovan
ADDF (ADF)^X	0110AA00	D	flyttalsaddition mellan celler och RØ och R1 resultat i RØ och R1
SUBF (SUF)^X	0110AA01	D	flyttalssubtraktion, analog med ADDF ovan
MPYF (MPF)^X	0110AA10	D	flyttalsmultiplikation, analog med ADDF
MPY (MPY)	0100AA10	D	heltalsmultiplikation mellan cell och RØ resultat i RØ
JMPR (JMR)	0011AA11	D	hopp, adress beroende på adresseringsmod
JMPI (JMI)	0011AA10	D	indirekt hopp, adress till cell med slutadress beror av adresseringsmod. Om moden är I erhålls sålunda indirekt indirekt hopp
JSRR (JSR)	0011AA01	D	subrutinhopp, analogt med JMPR ovan, återhoppsadress lagras automatiskt i SP
JSRI (JSI)	0011AA00	D	subrutinhopp, analogt med JMPI ovan
LOADI (LDI)	101000RR	I	ladda R_n direkt med I som tas med tecken
ADDI (ADI)	101001RR	I	addera direkt till R_n
LRA (LRA)	101010RR		kopierar RØ till R_n ,RØ bibehålls
LAR (LAR)	101011RR		kopierar R_n till RØ, R_n bibehålls
ADDR (ADR)	100010RR		R_n adderas till RØ, resultat i RØ
SUBR (SUR)	100011RR		subtraktion analog med ADDR ovan
AND (AND)	100000RR		logiskt OCH mellan R_n och RØ , resultat i RØ
CHS (CHS)	100001RR		teckenbyte i 2-komplement av register R_n
SHL (SHL)	100101RR	D	vänsterskift det antal steg som ges av D om D är negativ erhålls således högerskift skiftet är logiskt dvs utan teckenbevaring

ROR (ROR)	100111RR	D	rotera det antal steg åt höger som ges av D OBS D endast positivt dvs går ej att rotera åt vänster
ANDI (ANI)	111000RR	I	logiskt OCH direkt mellan I och R_n
JMP(Ø) (JMZ)	101100RR	I	vilkorligt hopp relativt PC (självrelativt) hoppet utförs om R_n är Ø , I med tecken
JMP(+) (JMG)	101101RR	I	hopp om R_n är större än eller lika med Ø
JMP(-) (JMN)	101110RR	I	hopp om R_n är mindre än Ø
JMP (JMP)	10111100	I	ovilkorligt självrelativt hopp
HLT (HLT)	00000001		haltinstruktion
NOP (NOP)	00000000		ingen operation, PC räknas upp med ett
CHSF (CSF)[x]	00000011		Byter tecken på flyttal i RØ och R1
RET (RET)	00000010		återhopp från subrutin
MPYI (MPI)	00001000	I	direkt heltalsmultiplikation mellan I och RØ I positivt tal mellan Ø och 127
MVG (MVG)[x]	00001010 AD2 AD3	AD1	flyttar en hel grupp av data från en plats i minnet till en annan . Instruktionen upptar tre ord
MVW (MVW)[x]	00001011 AD2 AD3	AD1	

x)Instruktioner som kan anses speciella för tillämpningen.

ASSEMBLER

Allmänt

När man konstruerar ett eget datorsystem,får man problem på grund av att mjukvaran,som finns tillgänglig till ett befintligt system,saknas.Ett stort problem är att alla program måste skrivas i maskinspråk,vilket går bra när man har korta program eller enkla funktioner,men ger stora problem vid större program.En av de första uppgifterna är därför att skaffa sig en assembler.

De krav man kan ställa på en assembler är följande:

1. Möjlighet att välja ut- och in-enheter
2. Klar och tydlig listning av källkod,positionsräknaren och binärkoden.
3. Sidindelning av listningen med programnamn,tid, datum och sidnummer.
4. Felmedelande
5. Utskrift av symboltabell och cross-referenser.

När man skriver en assembler kan man gå tillväga på två olika sätt:

1. Man kan skriva en assembler (i maskinspråk) på sin microdator.
2. Man kan utnyttja en befintlig assembler på en annan dator.

Metod 1 är mycket arbetskrävande då programmet blir stort och dessutom svårskrivet,då det måste skrivas i maskinspråk.Vi valde därför metod 2 med en PDP-11 som värddator.

PDP-11:s Assembly-Instruktioner

En assembly instruktion består av fyra fält:

LABEL: OPERATION OPERAND(ER) ;KOMMENTARER

Labelfältet

En label är en symbol,som tilldelas det nuvarande värdet av positionsräknaren.En label måste komma

först i instruktionen och måste avslutas med ett kolon(:).De första sex tecknen är signifikanta. Tillåtna tecken är bokstäverna A till Z,siffrorna 0 till 9 samt punkt och dollar.Emellertid är punkt och dollar reserverade för systemprogram.

Operationsfältet

Ett operationsfält kommer efter labelfältet i instruktionen och kan innehålla ett macro anrop,en instruktions memonamn eller ett assembler direktiv. Operationen kan föregås av ingen,en eller flera labels och kan följas av en eller flera operander, och/eller kommentarer.När operationen är ett memoma namn för en instruktion,specificeras en operationskod och vilken funktion,som de efterföljande operanderna skall ha.När operationen är ett assembly direktiv,så specificeras vissa funktioner,som skall utföras under assembleringen.Beträffande macro anrop återkommer jag senare.

Operandfältet

En oprand är den del av instruktionen på vilken operationen arbetar.Som operander kan användas uttryck, tal ,symboler eller macro argument.När flera operander ingår i en instruktion,separeras en från nästa genom ett av följande tecken:komma, "tab","space" eller vinkelperantes runt en eller flera operander.

Fält för kommentarer

Kommentarer börjar med ett semikolon och hela raden efter semikolonet ignoreras av assemblern.

Definiering av Macro

Kombinationer av assembly instruktioner,som återkommer ofta i programmet,kan med fördel bytas ut mot macro anrop,med eller utan operander.Macro anrop refererar till en macro definition,som i

sig innehåller assembly instruktioner med formella argument.När ett program assembleras,kontrollerar assemblern först om operationen är ett macro anrop, i så fall expanderas macron.Det är detta som gör att man kan använda MACRO 11 för att assemblera ett program skrivet i ett annat assembly språk.

Positionsräknaren

Det största problemet med att använda PDP-11 som värddator,var att MACRO 11 inkrementerar positionsräknaren med 2 för varje ord om 16 bitar i stället för att inkrementera med 1.Detta på grund av att PDP-11 är byte orienterad,vilket betyder att programräknaren inkrementeras med 1 för varje byte. Dessutom tolererar assemblern endast att 16 bitars instruktioner placeras ut med början på jämna positioner.Problemet löstes genom att i macro definitionerna multiplicera positionsräknaren med två innan instruktionen läggs ut i minnet,och därefter dividera positionsräknaren med två för att få fram det rätta värdet.
Detta medför att en begränsning införs i assemblern.Om man vill lägga ut ett tal i en cell,måste detta göras via ett macro,som vi kallat LAG,man skriver alltså t.ex.:

```
UT:    LAG     5000 ;HÄR LAGRAS ADRESSEN TILL UT
```

Macro Definitioner

Som synes i insruktionslistan,finns det två huvudtyper av instruktioner,de som har 16 bitars instruktionskod och de som har 48 bitars instruktionskod. Uppbyggnaden av macro definitionerna är emellertid lika,varför beskrivningen kan göras gemensam för bägge typerna.

Ett macro för instruktionen LOAD (LD) skulle kunna ha följande utseende:

```
.MACRO   LD
.=.*2
.WORD    <20!A!R>*400!<AD&377>
.=./2
.ENDM
```

Här betyder ! logiskt ELLER och & logiskt OCH. WORD direktivet används för att generera ord med data.OP delen multipliceras med 400(oktalt),vilket betyder skiftning åtta steg,och definierar på så sätt den övre byten.Adressdelen maskas med 377, vilket betyder att den definierar den undre byten. Vinkelparanteserna används för att få rätt värde på uttrycket,annars evalueras uttrycket strikt från vänster till höger.
Denna macro definition skulle emellertid medföra att positionsräknarens dubbla värde skulle listas och dessutom skulle WORD skrivas ut vilket inte är så vidare snyggt.
För att göra det enklare att införa instruktioner och för att spara minne,inför vi ett inre macro BODY 1,som får följande utseende:

```
.MACRO   .BODY 1    OP,AD
.NLIST   LOC
.NLIST   SRC
.=.*2
.WORD    OP*400!<AD&377>
.=./2
.LIST    SRC
.LIST    LOC
.ENDM
```

Här undertrycks utskrift av källkod och positionsräknaren i macrot av skäl som beskrivits ovan.

För att få en snygg listning,får vi då ge följande direktiv till assemblern i början av programmet:

```
.LIST     MEB,SRC,BIN,TTM
```

Dessa direktiv lagras i macrofilen,varför man inte behöver bry sig om det när man skriver program.
I macro definitionen för LOAD anropar man nu .BODY1 och definitionen får följande utseende:

```
.MACRO    LD          R,AD,A
.BODY1    20!A!R,AD
.ENDM
```

I anropet av .BODY1 används argumenten OP=20!A!R och AD=AD
Här är A lika med adresseringsmoden,som kan vara någon av följande:

S=0	Direkt adress
P=4	Självrelativ adress
R=10	Basrelativ adress
I=14	Indirekt adress

och R lika med registerval,som kan vara nogot av följande:

RØ=Ø
R1=1
R2=2
R3=3

OBS: Genom dessa definitioner blir S,P,R,I,RØ,R1,R2,R3och DA reserverade och får inte användas till något annat.
Denna definition av macro medför emellertid problem när man vill adressera självrelativt.Man måste nämligen då själv räkna ut avståndet,och kan inte använda metoden med labels,vilket är mycket enklare.Därför har vi lagt in .BODYØ,som endast kontrollerar om det är självrelativ adressering och i så fall räknar om adressen till rätt värde.Detta betyder att man även vid självrelativ adressering anger absolutadressen.
Det bör observeras att uttryck,som t.ex. .-2 , i adressfältet i princip är tillåtna men bör undvikas då punkten byter värde i .BODY1.

Det slutgiltiga macrot för LOAD blir alltså:

```
.MACRO    LD        R,AD,A
.BODYØ    20!A!R,AD,A
.ENDM
```

Det finns ytterligare ett inre macro: .BODY2 .Detta används när man direkt vill generera 16 bitars instruktionskod utan att gå via övre och undre byte,som man gör i .BODY1. På sid 34 och framåt finns alla macro definitionerna listade.

Och kom ihåg: " The assembler will only do what you write, not what you mean "

Införande av nya instruktioner

Hur man gör när man inför en ny instruktion visas enklast genom ett exempel.Antag att vi skall införa en ny instruktion,som heter ZER,som nollställer en cell i minnet.Operationskoden som skall genereras är: 1111AA11.
11110011=363(oktalt).Nu multipliceras emellertid OP med 400 i BODY 1 och om OP är större än 177 så överskrids talområdet,vi måste alltså göra om 363 till den negativa motsvarigheten.Vi antar då att den första biten är en teckenbit och får då i stället: 11110011⇒00001101 = 15.363 övergår alltså i -15.Definitionen får då följande utseende:

```
.MACRO   ZER     AD,A
.BODY Ø.  -15!A,AD,A
.ENDM
```

Här bör obseveras att det inte är nödvändigt att gå över från positivt till negativt tal,eftersom maskinen inte protesterar när talområdet överskrids, men det är nog säkrast så att man inte senare får fel,som ställer till trassel.

Kommando till assemblern

För assemblering i MACRO 11 behövs två filer : En fil INTEL.MAC med macro definitioner och en fil PGM.INT med program.
MACRO 11 anropas genom kommandot:

```
  R MACRO
```

och svarar med att skriva ut en stjärna.
Med kommandot:

```
PP:/E:ABS,CB:/C:S:E < DK:INTEL.MAC,PGM.INT
```

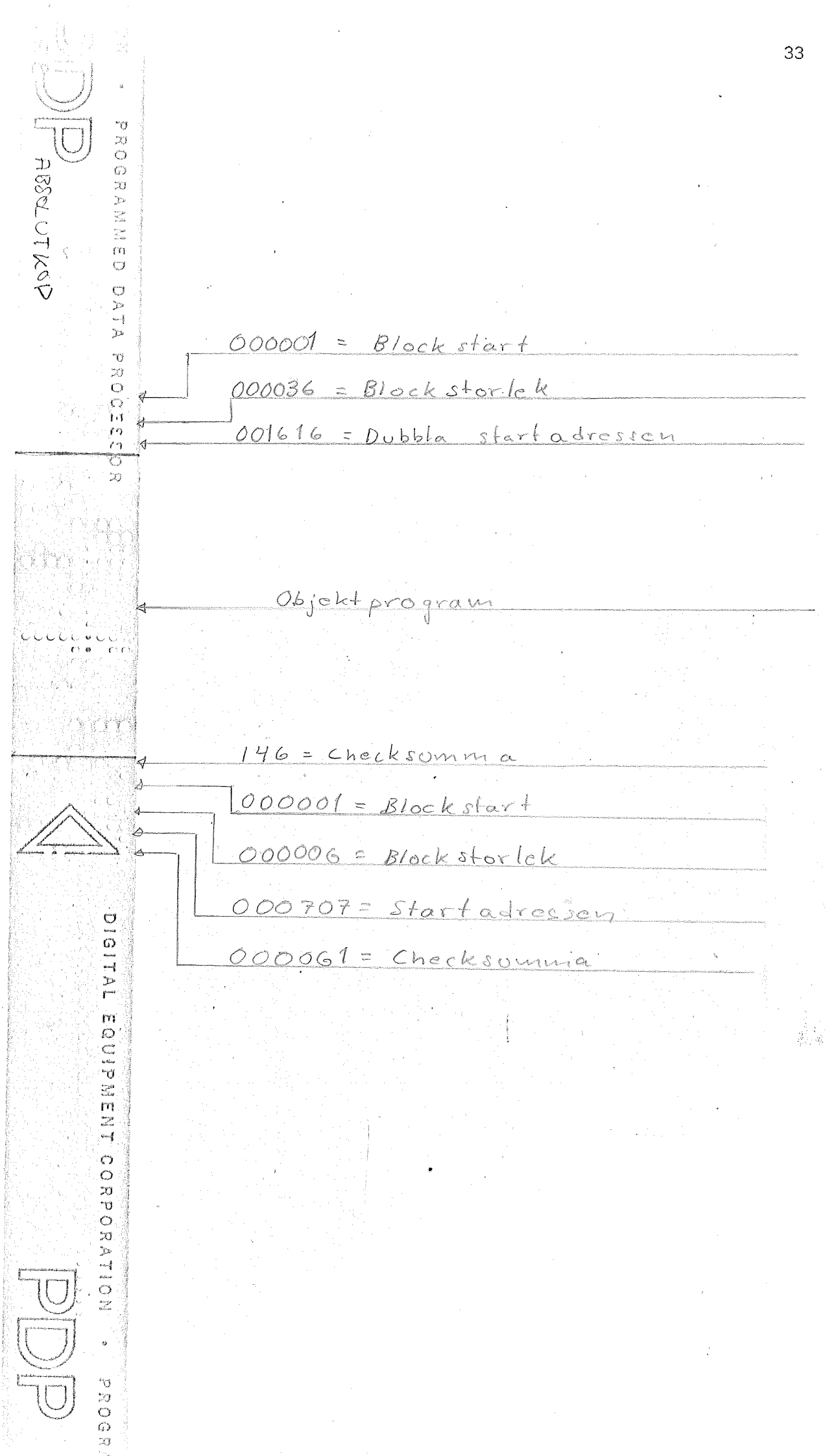
fås remsa med absolutkod på enhet PP samt listning på enhet CB (teletype).

Objektkodsremsan

Den remsa,som fås,innehåller två block.Det första blocket innehåller information om startadrees,programmets storlek samt objektkoden för programmet.I det andra blocket finns enbart information om startadressen,varför detta blocket inte används.

Block 1 inleds med en blockstart,därefter kommer blockstorleken,vilket är det totala antalet byte,som finns i blocket d.v.s. antalet byte i blockhuvudet plus antalet byte i propgrammet.Därefter kommer startadressen multiplicerad med 2. Multiplikationen med 2 beror på att positionsräknaren multipliseras med 2 i BODY.Sist i blocket kommer objektprogrammet, där varje 16 bits ord är 2 byte,med de minst signifikanta 8 bitarna först.Efter blocket följer en checksumma,som beräknas enligt följande:Varje byte i blocket antages vara ett tal mellan -200(oktalt) och +177(oktalt).Dessa adderas ihop.Summan manipuleras så att man får ett tal mellan -200 och +177 och adderas sedan till checksumman,varefter resultatet skall bli lika med noll.

Om programet,som skall assembleras är så stort att objektkoden består av mer än 40 byte,så delar assemblern upp det i delar om högst 40 byte.Objektkodsremsan kommer då att bestå av flera block av typ block 1 efter varandra,men endast ett block av typ block 2.


PROGRAMMED DATA PROCESSOR
000001 = Block start
000036 = Block storlek
001616 = Dubbla start adressen
Objekt program
146 = Checksumma
000001 = Block start
000006 = Block storlek
000707 = Start adressen
000061 = Checksumma
DIGITAL EQUIPMENT CORPORATION

```
.NLIST
.MACRO   LD          R,AD,A
.BODY0   20!A!R,AD,A
.ENDM
.MACRO   ST          R,AD,A
.BODY0   40!A!R,AD,A
.ENDM
.MACRO   ADM         R,AD,A
.BODY0   120!A!R,AD,A
.ENDM
.MACRO   SUM         R,AD,A
.BODY0   160!A!R,AD,A
.ENDM
.MACRO   LDF         AD,A
.BODY0   100!A,AD,A
.ENDM
.MACRO   STF         AD,A
.BODY0   101!A,AD,A
.ENDM
.MACRO   ADF         AD,A
.BODY0   140!A,AD,A
.ENDM
.MACRO   SUF         AD,A
.BODY0   141!A,AD,A
.ENDM
.MACRO   MPF         AD,A
.BODY0   142!A,AD,A
.ENDM
.MACRO   MPY         AD,A
.BODY0   102!A,AD,A
.ENDM
.MACRO   JMR         AD,A
.BODY0   63!A,AD,A
.ENDM
```

```
.MACRO   JMI        AD,A
.BODY0   62!A,AD,A
.ENDM
.MACRO   JSR        AD,A
.BODY0   61!A,AD,A
.ENDM
.MACRO   JSI        AD,A
.BODY0   60!A,AD,A
.ENDM
.MACRO   BCO        AD,A
.BODY0   103!A,AD,A
.ENDM
.MACRO   LDI        R,AD
.BODY1   -140!R,AD
.ENDM
.MACRO   ADI        R,AD
.BODY1   -134!R,AD
.ENDM
.MACRO   LRA        R
.BODY1   -130!R,0
.ENDM
.MACRO   LAR        R
.BODY1   -124!R,0
.ENDM
.MACRO   ADR        R
.BODY1   -170!R,0
.ENDM
.MACRO   SUR        R
.BODY1   -164!R,0
.ENDM
.MACRO   AND        R
.BODY1   -200!R,0
.ENDM
```

```
.MACRO   CHS       R

.BODY1   -174!R,0

.ENDM
.MACRO   SHL       R,AD

.BODY1   -154!R,AD

.ENDM

.MACRO   ROR       R,AD

.BODY1   -144!R,AD

.ENDM

.MACRO   ANI       R,AD

.BODY1   -40!R,AD

.ENDM

.MACRO   JMZ       R,AD
DA=AD-1-.

.BODY1   -120!R,DA

.ENDM

.MACRO   JMG       R,AD
DA=AD-1-.

.BODY1   -114!R,DA

.ENDM

.MACRO   JMN       R,AD
DA=AD-1-.

.BODY1   -110!R,DA

.ENDM

.MACRO   JMP       AD
DA=AD-1-.

.BODY1   -104,DA

.ENDM
.MACRO   MPI       AD

.BODY1   10,AD

.ENDM

.MACRO   DEN       AD

.BODY1   11,200+AD

.ENDM

.MACRO   NRM       AD

.BODY1   14+<AD/20>,200+AD
.ENDM
```

```
.MACRO   LAG      AD

.BODY2   AD

.ENDM

.MACRO   HLT

.BODY1   1,0

.ENDM

.MACRO   NOP

.BODY1   0,0

.ENDM

.MACRO   CSF

.BODY1   3,0

.ENDM

.MACRO   RET

.BODY1   2,0

.ENDM

.MACRO   CLF
.BODY1   17,0

.ENDM

.MACRO   SPZ
.BODY1   14,0

.ENDM

.MACRO   MVG      AD1,AD2,AD3

.BODY1   12,AD1

.BODY2   AD2

.BODY2   AD3

.ENDM
.MACRO   MVW      AD1,AD2,AD3

.BODY1   13,AD1

.BODY2   AD2

.BODY2   AD3

.ENDM

R0=0

R1=1

R2=2

R3=3
```

```
S=0

P=4

R=10

I=14
.MACRO   .BODY0   OP,AD,A
DA=AD
.IF      EQ       A-F
DA=AD-1-.
.ENDC
.BODY1   OP,DA
.ENDM

.MACRO   .BODY1   OP,AD

.NLIST   LOC

.NLIST   SRC

.=.*2

.WORD    OP*400!<AD&377>
.=./2

.LIST    SRC

.LIST    LOC

.ENDM

.MACRO   .BODY2   AD

.NLIST   LOC

.NLIST   SRC
.=.*2

.WORD    AD

.=./2

.LIST    SRC

.LIST    LOC

.ENDM

.ASECT

.LIST    MEB,SRC,BIN,TTM

.LIST
```

MONITOR

Vid konstruktion av mikrodatorsystem blir ett av problemen komunikationen med datorn. Hur data och program ska läsas in till maskinen samt hur resultat ska erhållas ut. Till maskinen i vårt aktuella fall finns ingen panel varifrån data kan knappas in, istället har en för ändamålet ganska lämplig bordskalkylator användts. Även denna metod var emellertid ett provisorium och ska ersättas av en bildskärm med tangentbord. För att denna ska kunna användas tillsammans med datorn måste någon typ av monitor konstrueras. För detta ändamål har vi skrivit ett monitorprogram med förjande rutiner:

LOAD rutin som läser in tal från tangentbord till adresserad cell i minnet.
Rutinen anropas enligt: L(OAD) startadressR
där tecknen inom parantes är frivilliga, startadress är adressen till den cell i minnet där lagringen ska börja. R betyder här "vagnretur" på bildskärmen. Monitorn svarar på anropet med att skriva ut aktuell adress till minnet avslutad med en blank. Därefter väntar monitorn på att talet ska skrivas på tangentbordet och avslutas med "R" .
Monitorn lagrar därefter in talet och skriver ut adressen till nästa cell i minnet och är åter redo att ta emot ett tal. För att komma ur LOAD-rutinen och återvända till monitorns väntetillstånd skrivs @ .

DUMP rutin som dumpar innehållet i konsekutiva minnesceller .
Rutinen anropas enligt: D(UMP) startadress stoppadressR
Monitorn svarar på komandot med att skriva ut adressen till minnescellen åtföljt av dess innehåll på en rad på bildskärmen och fortsätter med nästa cell på nästa rad tills stoppadressen är uppnådd. Därefter återvänder monitorn till sitt väntetillstånd.

RUN rutin som används när ett program i minnet ska exekveras .
Rutinen anropas med komandot: R(UN) startvärdeR
Innan monitorn lämnar över komandot till programmet som ska exekveras laddas begynnelsevärden till registren in från celler i SP, dessa celler benämns: RØR, R1R,R2R,R3R. Vilka i sin tur givits värden genom exempelvis LOAD-kommando.

REMSLOADER rutin som läser in program/data till minnet via en remsläsare.
Rutinen anropas med kommandot: P(REMSA) startadressR
Monitorn svarar med att styra remsläsaren och läsa in koden på remsan till minnet. Om startadressen i anropet sätts till Ø tas startadress från remsan, i annat fall den angivna startadressen. Programmet på remsan ska vara givet i absolutkod, åtta bitar utan paritetscheck, såsom det erhålls ut från en större maskin (ex PDP 11) på vilken programutveckling sker. Programmet på remsan föregås av en del information om startadress, blockstorlek och kontrollsumma. (se sid 32,33)

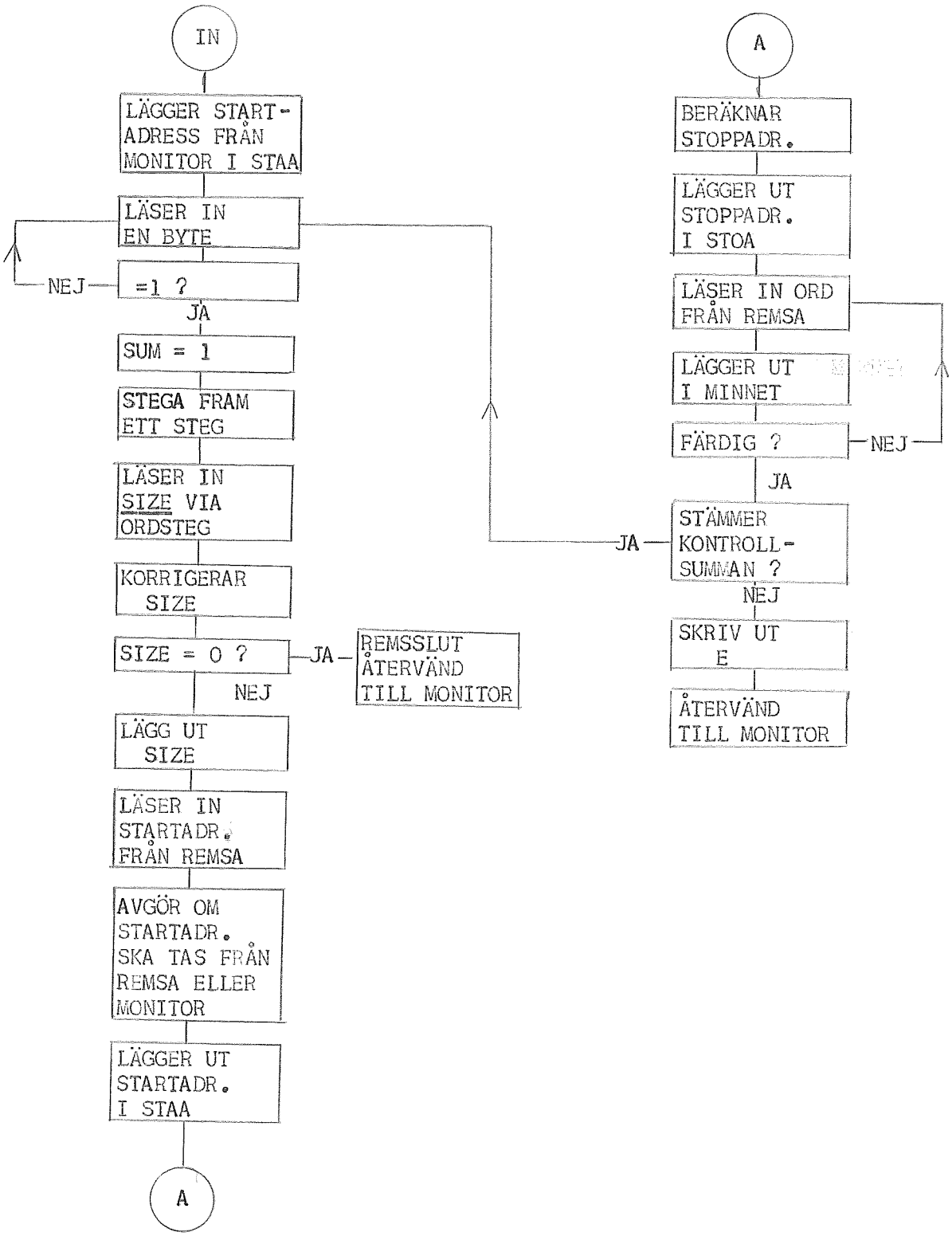
REMSLOADER (forts.)
Rutinen kontrollerar att den inlästa informationen ger en kontrollsumma som överensstämmer med den som är given på remsan. Om det härvid upptäcks ett fel svarar rutinen med att efter avslutad överföring av ett block skriva ut E> på bildskärmen.

Allmänt om monitorn
Alla adresser och all data monitorn tar emot och skriver ut är i oktal form. När monitorn går in i sitt väntetillstånd för nytt komando skrivs tecknet > ut på bildskärmen.
Om felaktigt kommando givits kan användaren upphäva verkan av detta genom att trycka på "rubout", härvid återgår monitorn i regel till sitt väntetillstånd. Undantag är om "rubout" används vid datainskrivning i LOAD-rutinen, då blir resultatet att monitorn på nytt skriver ut den aktuella adressen och data kan skrivas in.
Oberoende av vad datorn vid ett visst tillfälle är sysselsatt med kan monitorns väntetillstånd alltid nås om tangenten @ trycks ned. Detta kan vara lämpligt exempelvis om ett program har fastnat i en loop.
En fullständig, assemblerad,listning av monitorprogrammet återfinns på sid 41 och framåt, där återfinns även flödesschema över monitorns rutiner och tillhörande subrutiner.
Det ska här även framhållas att vi inte har haft möjlighet att testa monitorn på grund av att nödvändig hårdvara inte funnits tillgänglig. Detta medför naturligtvis att programmet säkerligen innehåller ett antal hittills oupptäckta fel.

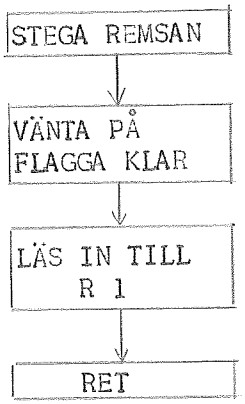
SAMMANFATTNING

Efter att ha arbetat med Intel 3000 har vi funnit att systemet är mycket flexibelt och dessutom snabbt.
Denna flexibilitet är emellertid inte så stor för en registermaskin,på grund av de begränsningar som hårdvaran utgör.I själva verket blir instruktionslistan för registermaskiner av samma typ som den för KMV-version 2,utom de markerade med x.Dessa instruktioner tar upp cirka 200 ord av microprogramminnet.Det vore önskvärt att leverantören tillhandahöll en sådan standardinstruktionslista.
Utöver dessa instruktioner kan användaren själv tillfoga special instruktioner.I detta fall är dessa bl.a. instruktioner för flytande aretmetik,som upptar ytterligare 256 ord.
Intresant ur reglerteknisk synpunkt vore att undersöka vilka instruktioner som skulle behövas för att underlätta implementering av reglersystem,samt hur mycket microprogram som skulle erfordras.
Till exempel en instruktion för beräkning av skalärprodukt.
En annan intresant vidare utveckling vore att undersöka hur en stackmaskin skulle kunna konstrueras med Intel 3000.
Det värdefullaste resultatet av arbetet torde dock vara framtagningen av cross-assemblerm,viken på ett helt avgörande sätt förenklar programeringsarbetet.
Det skall i sammanhanget ävet påpekas att den tillverkade cross--assemblern är avsedd för en 16 bitars målmaskin tillsammans med en PDP-11,som är byteorienterad.Den använda metoden kan emellertid direkt tillämpas på alla målmaskiner med vilken ordlängd som helst.
På samma sätt som cross-assemblern är utförd kan en motsvarande cross-microassembler tillverkas,en sådan kan vara till stor hjälp vid microprogrameringsarbetet.

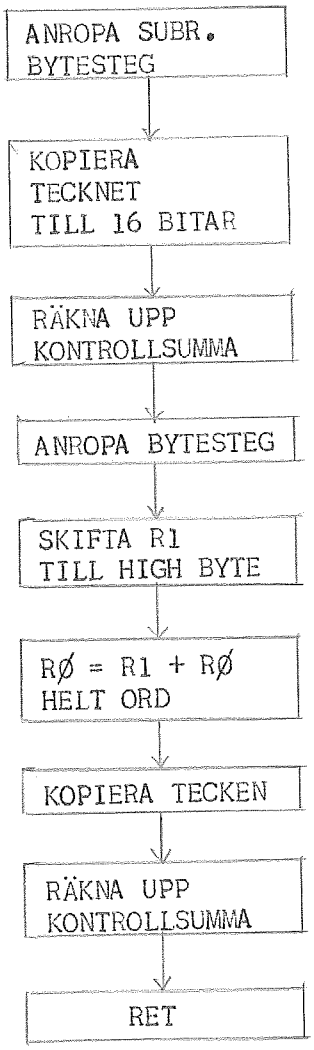
LOADER

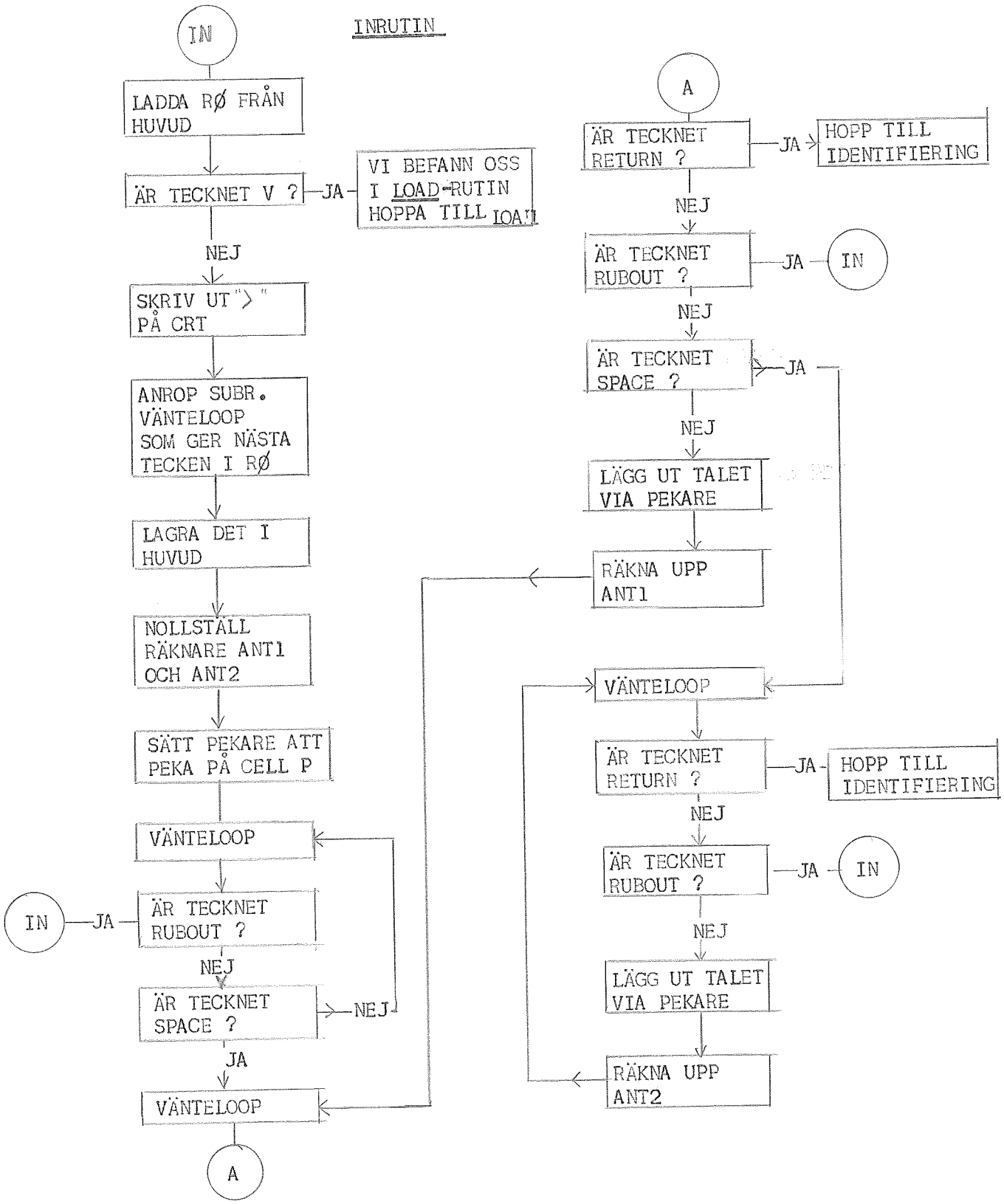

IN
LÄGGER START-
ADRESS FRÅN
MONITOR I STAA
LÄSER IN
EN BYTE
=1 ?
NEJ
JA
SUM = 1
STEGA FRAM
ETT STEG
LÄSER IN
SIZE VIA
ORDSTEG
KORRIGERAR
SIZE
SIZE = 0 ?
JA
REMSSLUT
ÅTERVÄND
TILL MONITOR
NEJ
LÄGG UT
SIZE
LÄSER IN
STARTADR.
FRÅN REMSA
AVGÖR OM
STARTADR.
SKA TAS FRÅN
REMSA ELLER
MONITOR
LÄGGER UT
STARTADR.
I STAA
A
A
BERÄKNAR
STOPPADR.
LÄGGER UT
STOPPADR.
I STOA
LÄSER IN ORD
FRÅN REMSA
LÄGGER UT
I MINNET
FÄRDIG ?
NEJ
JA
STÄMMER
KONTROLL-
SUMMAN ?
JA
NEJ
SKRIV UT
E
ÅTERVÄND
TILL MONITOR

SUBRUTIN BYTESTEG

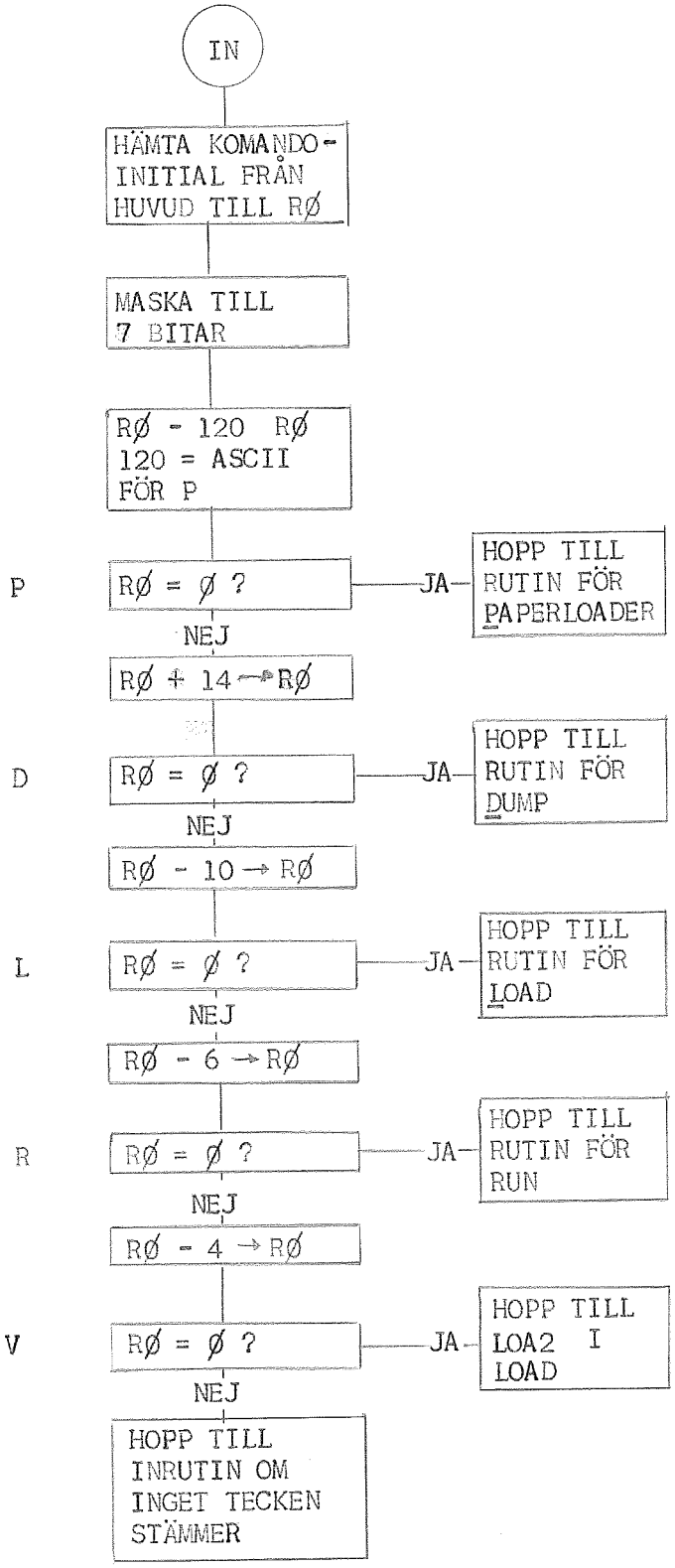


SUBRUTIN ORDSTEG

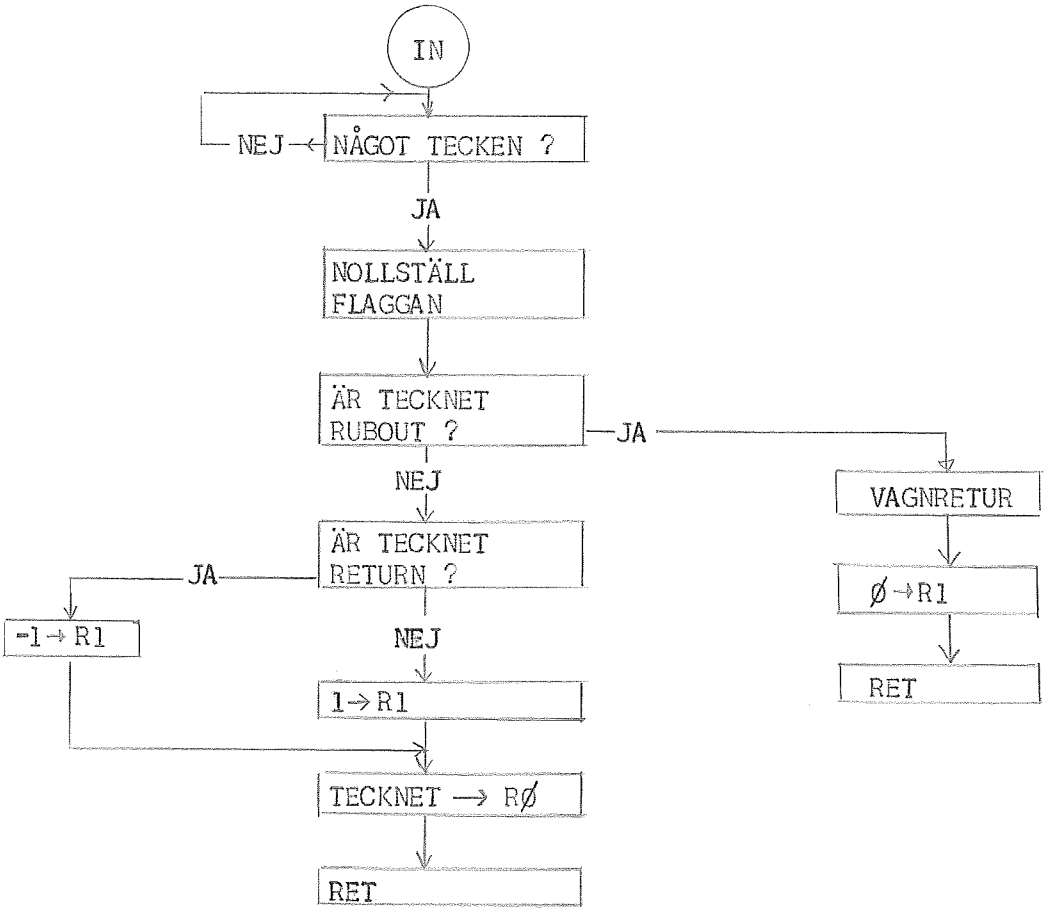



INRUTIN
IN
LADDA RØ FRÅN HUVUD
ÄR TECKNET V ?
JA
VI BEFANN OSS I LOAD-RUTIN HOPPA TILL LOA1
NEJ
SKRIV UT "> " PÅ CRT
ANROP SUBR. VÄNTELOOP SOM GER NÄSTA TECKEN I RØ
LAGRA DET I HUVUD
NOLLSTÄLL RÄKNARE ANT1 OCH ANT2
SÄTT PEKARE ATT PEKA PÅ CELL P
VÄNTELOOP
IN
JA
ÄR TECKNET RUBOUT ?
NEJ
ÄR TECKNET SPACE ?
NEJ
JA
VÄNTELOOP
A
A
ÄR TECKNET RETURN ?
JA
HOPP TILL IDENTIFIERING
NEJ
ÄR TECKNET RUBOUT ?
JA
IN
NEJ
ÄR TECKNET SPACE ?
JA
NEJ
LÄGG UT TALET VIA PEKARE
RÄKNA UPP ANT1
VÄNTELOOP
ÄR TECKNET RETURN ?
JA
HOPP TILL IDENTIFIERING
NEJ
ÄR TECKNET RUBOUT ?
JA
IN
NEJ
LÄGG UT TALET VIA PEKARE
RÄKNA UPP ANT2

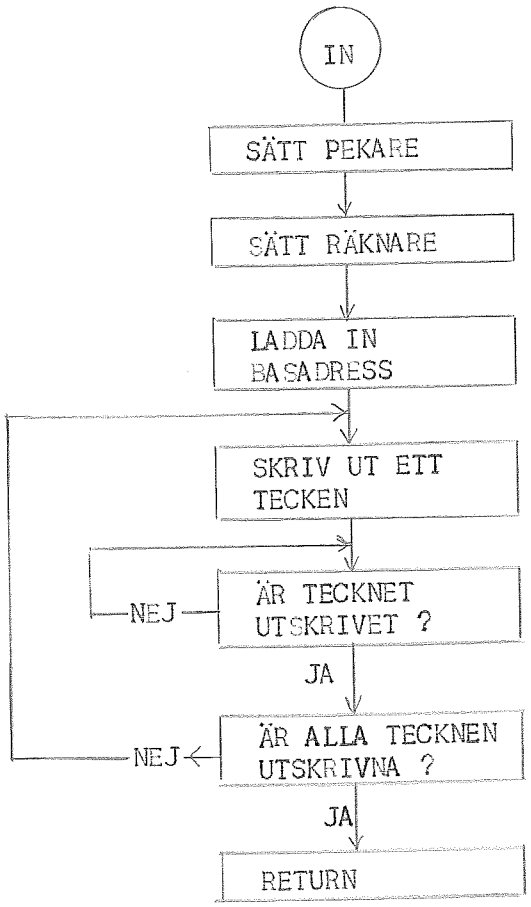
IDENTIFIERING AV KOMANDO



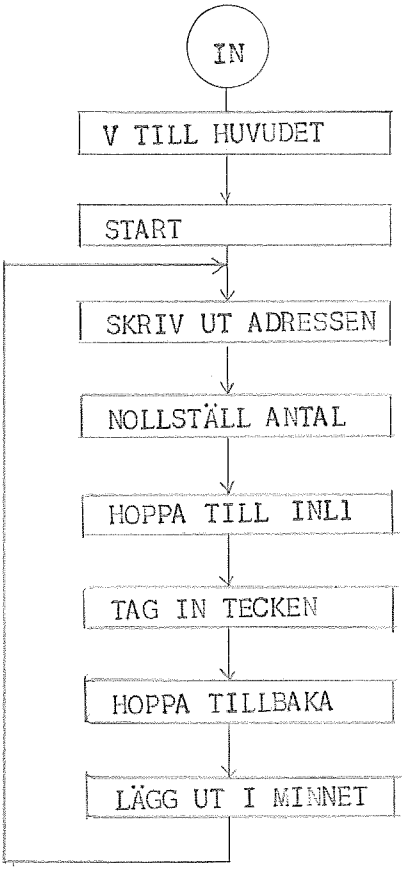
VÄNTELOOP


IN
NÅGOT TECKEN ?
NEJ
JA
NOLLSTÄLL FLAGGAN
ÄR TECKNET RUBOUT ?
JA
NEJ
VAGNRETUR
ØR1
RET
ÄR TECKNET RETURN ?
JA
-1R1
NEJ
1R1
TECKNET → RØ
RET

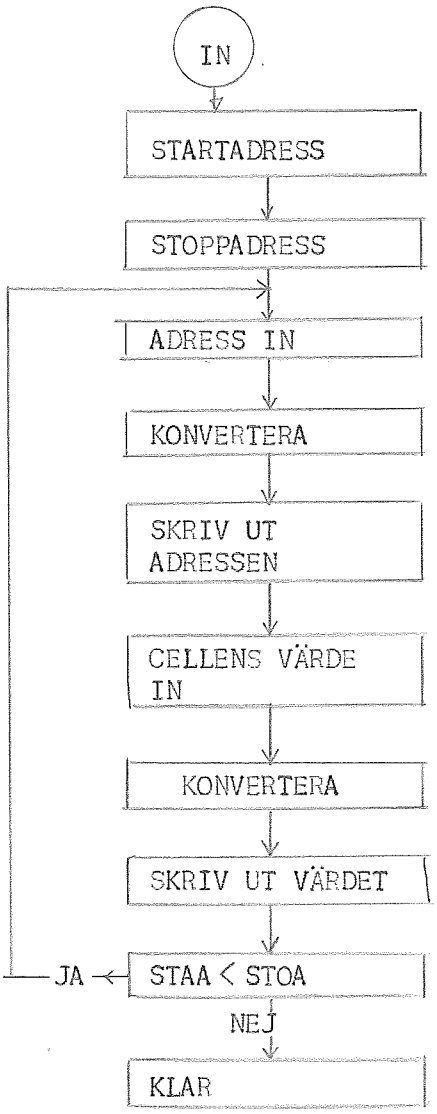
Subrutin OUT



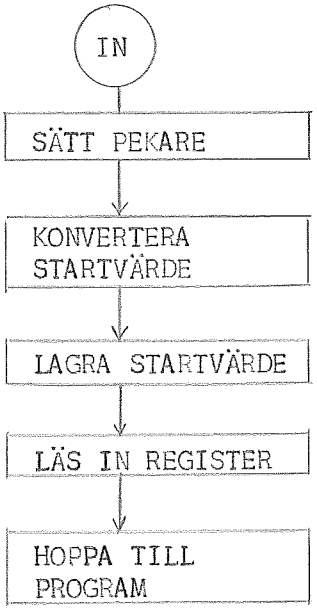
LOAD


IN
V TILL HUVUDET
START
SKRIV UT ADRESSEN
NOLLSTÄLL ANTAL
HOPPA TILL INL1
TAG IN TECKEN
HOPPA TILLBAKA
LÄGG UT I MINNET

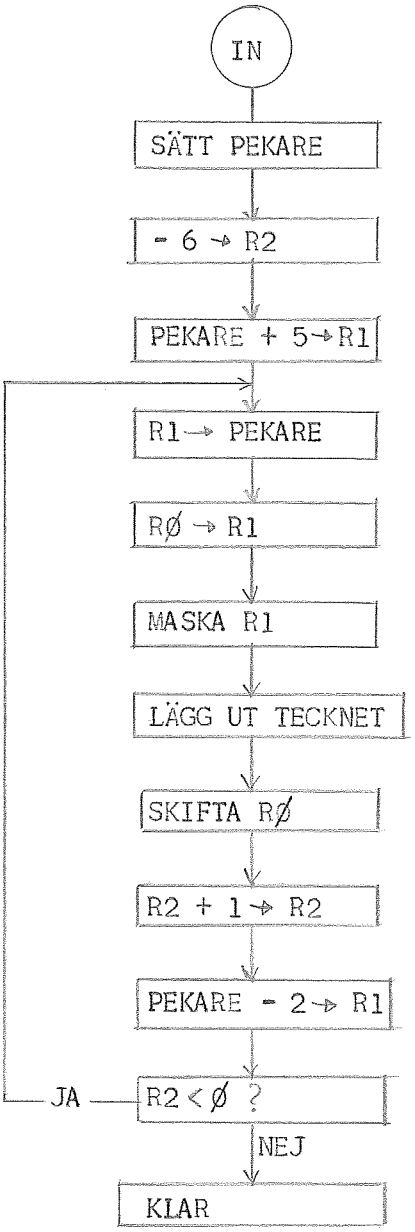
DUMP

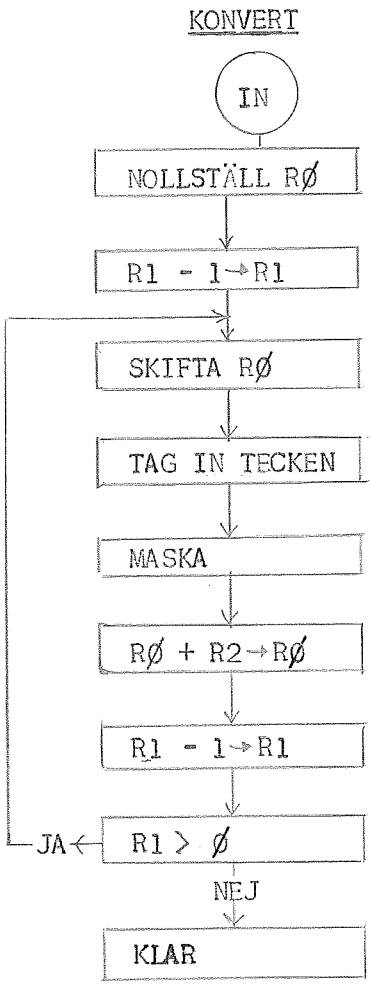

IN
STARTADRESS
STOPPADRESS
ADRESS IN
KONVERTERA
SKRIV UT ADRESSEN
CELLENS VÄRDE IN
KONVERTERA
SKRIV UT VÄRDET
JA
STAA < STOA
NEJ
KLAR

RUN


IN
SÄTT PEKARE
KONVERTERA STARTVÄRDE
LAGRA STARTVÄRDE
LÄS IN REGISTER
HOPPA TILL PROGRAM

BIKONVERT


IN
SÄTT PEKARE
- 6 → R2
PEKARE + 5→R1
R1→ PEKARE
RØ → R1
MASKA R1
LÄGG UT TECKNET
SKIFTA RØ
R2 + 1→ R2
PEKARE - 2→ R1
JA
R2 < Ø ?
NEJ
KLAR


KONVERT
IN
NOLLSTÄLL RØ
R1 - 1→R1
SKIFTA RØ
TAG IN TECKEN
MASKA
RØ + R2→RØ
R1 - 1→R1
JA
R1 > Ø
NEJ
KLAR

Funktion hos utnyttjade celler i SP

Adress Dec.	Adress Oktalt	Funktion
229	345	AKTE
230	346	RØR
231	347	R1R
232	350	R2R
233	351	R3R
234	352	ERROR
235	353	SIZE
236	354	HUVUD
237	355	PEK
238	356	Tecken buffert
239	357	
240	360	
241	361	
242	362	
243	363	
244	364	
245	365	
246	366	
247	367	
248	370	
249	371	
250	372	ANT1
251	373	ANT2
252	374	STAA
253	375	STOA
254	376	KFLAG
255	377	CFLAG

MONITOR MACRO-11 V3A000
TABLE OF CONTENTS

3- 1 BYTESTEG
3- 11 ORDSTEG
4- 1 ASKIFT
5- 1 INRUTIN
6- 1 IDENTIFIERING AV KOMANDO
7- 1 VANTELOOP
8- 1 PRINT
8- 12 OUT
9- 1 LOAD
10- 1 DUMP
11- 1 RUN
11- 13 REPEK
11- 20 STARTVARDE
12- 1 BIKONVERT
13- 1 KONVERT

```
        .TITLE MONITOR
;
; VERSION M001
;
;HANS HANSSON,INGVAR LARSSON
;  13/09/75
;
;MONITOR AR ETT PROGRAM FOR INLASNING OCH UTTESTNING AV
;PROGRAM,SKRIVNA FOR INTEL 3000,KMV VERSION 2
;
;
        SUM=345
        AKTE=345
        R0R=346
        R1R=347
        R2R=350
        R3R=351
        ERROR=352
        SIZE=353
        HUVUD=354
        PEK=355
        ANT1=372
        ANT2=373
        STAA=374
        STOA=375
        KFLAG=376
        CFLAG=377
        BYTE=0
        STEG=0
        CRT=0
        FLAGGA=0
        KEY=0
        CFT=0
```

```
        .SBTTL PAPERLOADER
;DETTA AR EN RUTIN SOM VIA REMSLASARE LASER IN ABSOLUTKOD TILL MINNE
;DEN ANROPAS FRAN MONITORN MED KOMANDOT P(REMSA) ADRE,
;OM ADR SATTS TILL 0 GES STARTADRESS AV REMSAN, OM ANNAN ADRESS
;ONSKAS ANGES DENNA I OKTAL FORM,
;RUTINEN BERAKNAR EN KONTROLLSUMMA, OM FEL UPPTACKS SVARAR MONITORN
;MED ATT EFTER AVSLUTAD OVERFARING SKRIVA UT E>
        .=1400
PLD0:   JSR     MEL0,P          ;SUBR. START GER STARTADRESS I STAA

PLD1:   JSR     BYT0,P          ;SUBR, BYTESTEG LASER IN EN BYTE TILL R1

        ADI     R1,-1           ;MINSKA VARDET AV INLAST TAL MED 1

        JMZ     R1,PLD2         ;OM DET INLASTA TALET AR 1 SKIPPA NASTA INST.

        JMP     PLD1            ;OM INTE HOPPA TILLBAKA OCH LAS NYTT

PLD2:   LDI     R2,1            ;

        ST      R2,SUM,S        ;LAGRA TALET 1 I CELL SUM

        JSR     BYT0,P          ;SUBR, BYTESTEG LASER IN NASTA BYTE

        JSR     ORD0,P          ;SUBR, ORDSTEG LASER IN ETT ORD (SIZE)

        ADI     R0,-6           ;KORRIGERAR SIZE

        JMZ     R0,INL0         ;OM REMSSLUT SA HOPPA TILL INLOOP

        SHL     R0,-1           ;DIVIDERA MED 2

        ST      R0,SIZE,S       ;LAGRAR STORLEK

        JSR     ORD0,P          ;SUBR,ORDSTEG LASER IN STARTADRESS

        LD      R1,STAA,S       ;INNEHALLET I STAA TILL R1

        JMZ     R1,PLD3         ;OM DETTA AR 0 TAS STARTADRESS FRAN REMSAN

        LDI     R2,-1           ;

        JMP     PLD4            ;I ANNAT FALL FRAN MONITORN ,HOPPA

PLD3:   LDI     R2,0            ;

        SHL     R0,-1           ;DUBBLA TILL ENKLA STARTADRESSEN

        LRA     R1              ;KOPIERA R0 TILL R1

        ST      R0,STAA,S       ;LAGRA STARTADRESS I STAA

PLD4:   ADM     R1,SIZE,S       ;ADDERA STORLEKEN TILL STARTADRESSEN
```

```
        ST      R1,STOA,S       ;SUMMAN GER STOPPADRESSEN
PLD7:   JSR     ORD0,P          ;SUBR,ORDSTEG LASER IN FORSTA ORDET AV PROGRAMMET
        ST      R0,STAA,I       ;SOM LAGRAS INDEREKT INKREMENT I STAA
        LD      R0,STAA,S
        SUM     R0,STOA,S       ;OM STARTADRESS-STOPPADRESS AR
        JMN     R0,PLD7         ;NEGATIV SKA VI FORTSATT ATT LASA IN
        LD      R1,SUM,S        ;HAMTA VARDET AV KONTROLLSUMMAN
        SHL     R1,10           ;SKIFTEN GORS FOR ATT SUM SKA INNEHALLA DE LAGRE
        JSR     ASK0,P          ;8 BITAR I LIKHET MED KONTROLLSUMMAN
        LAR     R1              ;KOPIERA R1 TILL R0 DVS SUM TILL R0
        JSR     BYT0,P          ;LASER IN BYTE SOM AR KONTROLLSUMMAN
        SHL     R1,10           ;SKIFTAR FOR ATT KOPIERA TECKNET
        JSR     ASK0,P
        ADR     R1              ;R0+R1 TILL R0
        JMZ     R0,PLD5         ;TEST OM KONTROLLSUMMA STAMMER
        LDI     R0,-73          ;OM INTE
        JSR     PRT0,P          ;PRINT
        JMP     INL0            ;OM FEL SA HOPPA TILL INLOOP
PLD5:   JMN     R2,PLD6         ;OM STARTADRESS FRAN MONITORN SA FORTSATT
        ST      R2,STAA,S       ;ANNARS NOLLSTALL STAA
PLD6:   JMP     PLD1            ;HOPP TILL MONITORNS PLD1
```

```
        .SBTTL BYTESTEG
;SUBRUTIN BYTESTEG STEGAR FRAM REMSAN ETT STEG , VANTAR TILLS
;REMSLASAREN AR KLAR VAREFTER INLASNING AV BYTE TILL R1  SKER
BYT0:   LD      R3,UT,P         ;LADDAR BASREGISTRET

        ST      R0,STEG,R       ;STEGAR FRAM REMAAN ETT STEG

        LD      R3,IN,P

BYT1:   LD      R1,FLAGGA,R     ;KONTROLERAR OM REMSLASAREN AR KLAR

        JMZ     R1,BYT1         ;OM INTE KOLLA IGEN

        LD      R1,BYTE,R       ;OM KLAR , HAMTA BYTE TILL R1

        RET

        .SBTTL ORDSTEG
        ;SUBRUTINEN ORDSTEG LASER IN TVA BYTE , SKIFTAR IHOP DEM TILL
;ETT ORD SAMT RAKNAR UPP KONTROLLSUMMAN
ORD0:   JSR     BYT0,P          ;SUBR, BYTESTEG FOR INLASNING AV LOW BYTE

        LAR     R1              ;R1 TILL R0

        SHL     R1,10           ;SKIFTAR FOR ATT KOPIERA TECKEN

        JSR     ASK0,P          ;ASKIFT

        ADM     R1,SUM,S        ;VARDET AV INLAST BYTE ADDERAS TILL SUM

        ST      R1,SUM,S        ;SUM LAGRAS UT

        JSR     BYT0,P          ;SUBR, BYTESTEG FOR INLASNING AV HIGH BYTE

        SHL     R1,10           ;

        ADR     R1              ;HIGH BYTE ADDERAS TILL LOW BYTE OCH GER ORDET

        JSR     ASK0,P          ;HIGH BYTE SKIFTAS FOR ATT KOPIERA TECKNET

        ADM     R1,SUM,S        ;KONTROLLSUMMAN RAKNAS UPP

        ST      R1,SUM,S        ;OCH LAGRAS UT

        RET
```

```
        ,SBTTL ASKIFT
;SUBRUTIN SOM SKIFTAR ETT ORD ARITMETRISKT 8 STEG AT HOGER
ASK0:   JMG     R1,ASK1         ;OM R1 AR NEGATIV

        ADM     R1,ASK2,P       ;SA ADDERA FOR ATT KOPIERA TECKENBITEN

ASK1:   ROR     R1,10           ;ROTERA

        RET

ASK2:   LAG     377

UT:     LAG     50000           ;BASADRESS TILL UTENHETER

MEL0:   JMP     STA0            ;MELLANLANDNING

IN:     LAG     30000           ;BASADRESS TILL INENHETER
```

```
        .SBTTL INRUTIN
;PROGRAM SOM SKOTER OM INMATNING AV KOMANDO OCH TAL FRAN KEY-
;BOARD
INL0:   LD      R0,HUVUD,S      ;HUVUDET TILL R0
        ADI     R0,-126         ;ADDERA ASCII FOR V
        JMZ     R0,LOA1         ;OM TECKNET AR LIKA MED V SA HOPPA TILL DATIN
        LDI     R0,-102         ;ASCII FOR > TILL R0
        JSR     PRT0,P          ;PRINT
        JSR     VTL0,P          ;VANTELOOP
        ST      R0,HUVUD,S      ;LAGRA KOMANDOT
        LDI     R0,0            ;0 TILL R0
        ST      R0,ANT1,S       ;NOLLSTALL ANT1
        ST      R0,ANT2,S       ;NOLLSTALL ANT2
        JSR     REP0,P          ;SATT PEKAREN
INL2:   JSR     VTL0,P          ;VANTELOOP
        JMZ     R1,INL0         ;OM R1 AR NOLL SA HOPPA TILL BORJAN
        ADI     R0,-40          ;ADDERA ASCII FOR SPACE
        JMZ     R0,INL1         ;OM DET VAR SPACE SA SKIPPA NASTA INSTRUKTION
        JMP     INL2            ;HOPPA TILLBAKA I VANTAN PA SPACE
INL1:   JSR     VTL0,P          ;VANTELOOP
        JMN     R1,IDE0         ;OM R1 VAR NEGATIV SA HOPPA TILL IDENTIFIERING
        JMZ     R1,INL0         ;OM R1 VAR NOLL SA HOPPA TILL BORJAN
        ADI     R0,-40          ;ADDERA ASCII FOR SPACE
        JMZ     R0,INL3         ;OM DET VAR SPACE SA HOPPA TILL INL3
        ST      R0,PEK,I        ;LAGG UT SIFFRAN
        LD      R0,ANT1,I       ;OKA ANT1 MED 1
        JMP     INL1            ;HOPPA TILL INL1
INL3:   JSR     VTL0,P          ;VANTELOOP
```

```
JMN     R1,IDE0         ;OM R1 AR NEGATIV SA HOPPA TILL IDENTIFIERING
JMZ     R1,INL0         ;OM R1 AR NOLL SA HOPPA TILL BORJAN
ST      R0,PEK,I        ;LAGG UT SIFFRAN
LD      R0,ANT2,I       ;OKA ANT2 MED 1
JMP     INL3            ;HOPPA TILLBAKA TILL INL3
```

```
        .SBTTL IDENTIFIERING AV KOMANDO
;PROGRAM SOM IDENTIFIERAR DET KOMANDO SOM GES FRAN KEYBOARD OCH
;GER HOPP TILL RATT RUTIN
IDE0:   LD      R0,HUVUD,S      ;HUVUDET TILL R0
        JSR     REP0,P          ;SATT PEKAREN
        ANI     R0,177          ;MASKA
        ADI     R0,-120         ;ADDERA ASCII FOR P
        JMZ     R0,PLD0         ;OM R0 AR NOLL SA HOPPA TILL PAPERLOADER
        ADI     R0,14           ;ADDERA FOR ATT TESTA OM D
        JMZ     R0,DMP0         ;OM R0 AR NOLL SA HOPPA TILL DUMP
        ADI     R0,-10          ;ADDERA FOR ATT TESTA OM L
        JMZ     R0,LOA0         ;OM R0 AR NOLL SA HOPPA TILL LOAD
        ADI     R0,-6           ;ADDERA FOR ATT TESTA OM R
        JMZ     R0,RUN0         ;OM R0 AR NOLL SA HOPPA TILL RUN
        ADI     R0,-4           ;ADDERA FOR ATT TESTA OM V
        JMZ     R0,LOA2         ;OM R0 AR NOLL SA HOPPA TILL LOA2
        JMP     INL0            ;HOPPA TILL INRUTIN
```

```
          .SBTTL VANTELOOP
;SUBRUTIN FOR ATT VANTA PA TECKEN OCH FOR ATT TA HAND OM RUBOUT
;OCH RETURN. SUBRUTINEN ANVANDER SAMTLIGA REGISTER.
VTL0:     LD      R0,KFLAG,S      ;FLAGGAN TILL R0

          JMN     R0,VTL0         ;NAGOT TECKEN?

          LDI     R0,0            ;JA

          ST      R0,KFLAG,S      ;NOLLSTALL FLAGGAN

          LD      R0,AKTE,S       ;TECKEN IN

          ANI     R0,177          ;MASKA

          ADI     R0,-177         ;VILKET TECKEN?

          JMZ     R0,VTL1         ;RUBOUT

          JMP     VTL2

VTL1:     LDI     R0,-163         ;ASCII FOR RETURN TILL R0

          JSR     PRT0,P          ;PRINT

          LDI     R1,0            ;0 TILL R1

          RET

VTL2:     ADI     R0,177          ;ATERSTALL

          ADI     R0,-15          ;VILKET TECKEN

          LDI     R1,-1           ;-1 TILL R1

          JMZ     R0,VTL3         ;OM DET VAR RETURN SA HOPPA TILLBAKA

          LDI     R1,1            ;1 TILL R1

          ADI     R0,15           ;ATERSTALL

VTL3:     RET
```

```
        .SBTTL PRINT
;SUBRUTIN FOR ATT SKRIVA UT ETT TECKEN LAGRAT I R0
;PA CRT;N.DESSUTOM ANVANDS REGISTER R3
PRT0:   LD      R3,UT,P         ;LADDA BASREGISTRET

        ST      R0,CRT,R        ;SKRIV UT TECKNET

        LDI     R0,-1           ;-1 TILL R0

        ST      R0,CFLAG,S      ;R0 TILL CFLAG

PRT1:   LD      R0,CFLAG,S      ;CFLAG TILL R0

        JMN     R0,PRT1         ;TECKNET UTSKRIVET?

        RET                     ;JA

        NOP

        .SBTTL OUT
;SUBRUTIN SOM SKRIVER UT TECKEN , SOM FINNS LAGRADE I KONSEKUTIVA
;CELLER,MED BORJAN I CELL P, PA CRT,RUTINEN KOMUNISERAR MED
;AVBROTTSRUTINEN VIA EN CELL SOM HETER CFLAG.
OUT0:   LDI     R1,-7           ;-7 TILL R1=RAKNARE

        JSR     REP0,P          ;SATT PEKAREN

        LDI     R0,-140         ;"SPACE" TILL R1

        ST      R0,PEK+7,S      ;"SPACE" TILL (P+6)

OUT1:   LD      R0,PEK,I        ;TECKEN TILL R0

        ADI     R1,1            ;RAKNA UPP RAKNAREN

        JSR     PRT0,P          ;PRINT

        JMN     R1,OUT1         ; AR ALLA TECKNEN UTSKRIVNA?

        RET                     ;JA
```

```
        .SBTTL LOAD
;PROGRAM FOR ATT LADDA IN TAL FRAN KEYBOARD TILL MINNET
LOA0:   LDI     R0,126          ;ASCII FOR V TILL R0

        ST      R0,HUVUD,S      ;R0 TILL HUVUDET

        JSR     STA0,P          ;STARTVARDE

LOA1:   LD      R0,STAA,S       ;STARTSDRESS TILL R0

        JSR     BIK0,P          ;BIKONVERTERA

        JSR     OUT0,P          ;SKRIV UT

        LDI     R0,0            ;NOLLSTALL R0

        ST      R0,ANT1,S       ;R0 TILL ANT1

        JMP     INL1            ;HOPPA TILL INPUT

LOA2:   LD      R1,ANT1,S       ;ANTAL TECKEN TILL R1

        JSR     KON0,P          ;KONVERTERA

        ST      R0,STAA,I       ;LAGG UT I MINNET

        JMP     LOA1            ;HOPPA TILL LOA1
```

```
        .SBTTL DUMP
;PROGRAM FOR ATT DUMPA INNEHALLET I MINNESCELLER,
;START- OCH STOPP-ADRESS FINNS LAGRADE I KONSEKUTIVA
;CELLER MED BORJAN I CELL P,ANTAL TECKEN FINNS I ANT1 OCH ANT2
DMP0:   JSR     STA0,P          ;STARTVARDE

        LD      R1,ANT2,S       ;ANTAL TILL R1

        JSR     KON0,P          ;KONVERTERA

        ST      R0,STOA,S       ;LAGRA STOPPADRESS

DMP1:   LD      R0,STAA,S       ;ADRESSEN IN

        JSR     BIK0,P          ;BIKONVERT

        JSR     OUT0,P          ;SKRIV UT

        LD      R0,STAA,I       ;INNEHALL IN

        JSR     BIK0,P          ;BIKONVERT

        JSR     OUT0,P          ;SKRIV UT

        LDI     R0,-163         ;ASCII FOR RETURN TILL R0

        JSR     PRT0,P          ;PRINT

        LD      R0,STAA,S       ;STOPPADRESS IN

        SUM     R0,STOA,S       ;- AKTUELL ADRESS

        JMG     R0,DMP1         ;KLAR ?

        JMP     INL0            ;JA,HOPPA TILL INLOOP
```

```
        .SBTTL RUN
;PROGRAM FOR ATT TESTA ANDRA PROGRAM,REGISTER VARDENA
;LASES IN FRAN SP TILL REGISTERNA INNAN KONTROLLEN LAMNAS OVER
;TILL DET PROGRAM SOM SKALL TESTAS,STARTADRESSEN FINNS I KON-
;SEKUTIVA CELLER MED BORJAN I CELL P,ANTAL TECKEN FINNS I
;ANT1
RUN0:   JSR     STA0,P          ;STARTVARDE

        LD      R0,R0R,S        ;R0R TILL R0

        LD      R1,R1R,S        ;R1R TILL R1

        LD      R2,R2R,S        ;R2R TILL R2

        LD      R3,R3R,S        ;R3R TILL R3

        JMR     STAA,I          ;HOPPA TILL PROGRAM

        .SBTTL REPEK
;SUBRUTIN SOM SATTER STARTVARDE FOR PEKAREN,RUTINEN
;ANVANDER REGISTER R3
REP0:   LDI     R3,167          ;167 TILL R3

        SHL     R3,1            ;2*R3 TILL R3

        ST      R3,PEK,S        ;LAGRA VARDET I PEKAREN

        RET

        .SBTTL STARTVARDE
;SUBRUTIN FOR ATT TRANSFORMERA STARTADRESSEN FRAN OKTALA
;TECKEN,LAGRADE I KONSEKUTIVA CELLER MED BORJAN I CELL P
;TILL ETT BINART TAL,SOM LAGRAS I CELL STAA
STA0:   JSR     REP0,P          ;SATT PEKAREN

        LD      R1,ANT1,S       ;ANTAL TILL R1

        JSR     KON0,P          ;KONVERTERA

        ST      R0,STAA,S       ;LAGRA STARTVARDE

        RET
```

```
        .SBTTL BIKONVERT
;SUBRUTIN FOR KONVERTERING AV TAL FRAN BINAR KOD
;TILL OKTAL KOD.INVARDET FINNS I R0 OCH UTVARDENA
;LAGRAS I KONSEKUTIVA CELLER MED BORJAN I CELL P
;RUTINEN UTNYTTJAR DESSUTOM REGISTERNA R1 OCH R2
BIK0:   JSR     REP0,P          ;SATT PEKAREN

        LDI     R2,-6           ;-6 TILL R2

        LD      R1,PEK,S        ;PEKAREN TILL R1

        ADI     R1,5            ;R1+5 TILL R1

        ST      R1,PEK,S        ;R1 TILL PEKAREN

BIK1:   LRA     R1              ;R0 TILL R1

        ANI     R1,7            ;MASKA R1

        ADI     R1,-160         ;OKTALT TILL ASCII

        ST      R1,PEK,I        ;LAGG UT TECKNET

        SHL     R0,-3           ;SKIFTA R0

        ADI     R2,1            ;OKA R2 MED 1

        LD      R1,PEK,S        ;PEKAREN TILL R1

        ADI     R1,-2           ;MINSKA R1 MED 2

        JMN     R2,BIK1         ;KLAR ?

        RET                     ;JA
```

```
        ,SBTTL KONVERT
;SUBRUTIN FOR ATT KONVERTERA TAL FRAN OKTAL KOD
;TILL BINAR KOD,INVARDENA FINNS LAGRADE I KONSEKUTIVA
;CELLER MED ADRESSEN TILL FORSTA VARDET I PEKAREN,
;RESULTATET LAGRAS I R0,INFORMATION OM ANTALET TECKEN
;FINNS I R1
KON0:   LDI     R0,0            ;NOLLSTALL R0

KON1:   ADI     R1,-1           ;MINSKA R1 MED 1

        SHL     R0,3            ;SKIFTA R0 3 STEG VANSTER

        LD      R2,PEK,I        ;TAG IN TECKNET

        ANI     R2,7            ;MASKA R2

        ADR     R2              ;ADDERA R2 TILL R0

        ADI     R1,-1           ;MINSKA R1 MED 1

        JMG     R1,KON1         ;KLAR?

        RET                     ;JA

        ,END
```

MONITOR MACRO-11 V3A000 PAGE 13+
SYMBOL TABLE

```
AKTE   = 000345      ANT1   = 000372      ANT2   = 000373
ASK0     001500      ASK1     001502      ASK2     001504
BIK0     001705      BIK1     001712      BYTE   = 000000
BYT0     001454      BYT1     001457      CFLAG  = 000377
CRT    = 000000      CRT    = 000000      DA     = 177771
DMP0     001646      DMP1     001652      ERROR  = 000352
FLAGGA = 000000      HUVUD  = 000354      I      = 000014
IDE0     001546      IN       001507      INL0     001510
INL1     001530      INL2     001523      INL3     001540
KEY    = 000000      KFLAG  = 000376      KON0     001724
KON1     001725      LOA0     001631      LOA1     001634
LOA2     001642      MEL0     001506      ORD0     001463
OUT0     001620      OUT1     001624      P      = 000004
PEK    = 000355      PLD0     001400      PLD1     001401
PLD2     001405      PLD3     001422      PLD4     001426
PLD5     001451      PLD6     001453      PLD7     001430
PRT0     001610      PRT1     001614      R      = 000010
REP0     001674      RUN0     001666      R0     = 000000
RUR    = 000346      R1     = 000001      R1R    = 000347
R2     = 000002      R2R    = 000350      R3     = 000003
R3R    = 000351      S      = 000000      SIZE   = 000353
STAA   = 000374      STA0     001700      STEG   = 000000
STOA   = 000375      SUM    = 000345      UT       001505
VIL0     001564      VIL1     001575      VTL2     001601
VIL3     001607
```

```
          .SBTTL AVBROTTSRUTINER
;
;FOR KEYBOARD
;
          ST        R0,R0R,S          ;R0 TILL R0R
          ST        R3,R3R,S          ;R3 TILL R3R
          LD        R3,IN,P           ;BASADRESS IN
          LD        R0,KEY,R          ;KEYBOARD TILL R0
          ST        R0,AKTE,S         ;R0 TILL AKTE
          ANI       R0,177            ;MASKA
          ADI       R0,-100           ;ADDERA ASCII FOR ALFA
          JMZ       R0,ALFA           ;ALFA ?
          LDI       R0,1              ;NEJ,1 IL R0
          ST        R0,KFLAG,S        ;R0 TILL KFLAG
          LD        R0,R0R,S          ;R0R TILL R0
          LD        R3,R3R,S          ;R3R TILL R3
          RIT
ALFA:     ION                         ;INTERUPT ON
          ST        R0,HUVUD,S        ;NOLLSTALL HUVUDET
          JMI       7,P               ;HOPPA TILL INLOOP
;
;FOR CRT:N
;
          ST        R3,R3R,S          ;R3 TILL R3R
          LDI       R3,0              ;NOLL TILL R3
          ST        R3,CFLAG,S        ;R3 TILL CFLAG
          LD        R3,UT,P           ;BASADRESS IN
          ST        R0,CFT,S          ;ATERSTALL CRT FLAGGAN
          LD        R3,R3R,S          ;R3R TILL R3
          RIT
AVB0:     LAG       1510              ;ADRESS FOR INLOOP
UT:       LAG       50000             ;BASADRESS FOR UTENHETER
IN:       LAG       30000             ;BASADRESS FOR INENHETER
```

OPERATIONSTIDER

Operationstider för KMV-version 2,angivna i µS.Vid självrelativ och basrelativ adressering tilkommer 0.45µS.Vid indirekt adressering tillkommer 0.9µS.

LD	1.35	SUR	1.2
ST	1.05	AND	0.9
LDF	1.5	CHS	0.6
STF	1.2	SHL	1.05 + 0.3N[1)]
ADM	1.35	ROR	1.35 + 0.3N[1)]
SUM	1.5	ANI	0.45
ADF	7-25	JMZ	1.65
SUF	7-25	JMG	1.65
MPF	40-45	JMN	1.65
MPY	15-20	JMP	1.05
JMR	1.65	HLT	0.6
JMI	1.65	NOP	0.6
JSR	1.65	CSF	0.6
JSI	1.65	RET	1.2
LDI	0.9	MPI	15-20
ADI	1.05	MVG	1.8 + 0.45M[2)]
LRA	1.2	MVW	1.8 + 0.45M[2)]
LAR	1.05		
ADR	1.05		

1) N betyder antal steg som skall skiftas eller roteras.

2) M betyder antal ord som skall flyttas.

```
                                ; INGVAR LARSSON,HANS HANSSON
                                ; DEMONSTRATIONSPROGRAM FOR CROSS-ASSEMBLER
                                ;TILL KMV-VERSION 2
                                ;
                                ; PROGRAMMET ADDERAR 5 TAL I EN VEKTOR,
                                ; PEKAREN TILL FORSTA ELEMENTET FINNS I CELL 100
                                ; RESULTATET MULTIPLICERAS MED ETT TAL I
                                ; KONSTANTMINNET,BASADRESSEN DIT FINNS I R3,
                                ; SLUTRESULTATET LAGGS UT I CELL 200
                                ;
                        000100          PEK=100
                        000052          PLATS=52
                        000200          RESULT=200
000000                          BEG:    LD      R0,PEK,I        ;FORSTA ELEMENTET TILL R0
016100
000001                                  LDI     R1,3            ;3 TILL R1 (RAKNARE)
120403
000002                          LOOP:   ADM     R0,PEK,I        ;ELEMENT ADDERAS TILL R0
056100
000003                                  ADI     R1,-1           ;MINSKA RAKNAREN
122777
000004                                  JMG     R1,LOOP         ;AR RAKNAREN MINDRE AN 0
132775
000005                                  MPY     PLATS,R         ;JA,MULTIPLICERA
045052
000006                                  ST      R0,RESULT,S     ;LAGG UT RESULTATET
020200
                        000001          .END
```

SYMBOL TABLE

```
BEG     000000          DA    = 000200          I      = 000014
LOOP    000002          P     = 000004          PEK    = 000100
PLATS = 000052          R     = 000010          RESULT= 000200
R0    = 000000          R1    = 000001          R2     = 000002
R3    = 000003          S     = 000000
```