CODEN:LUTFD2/(TFRT-5256)/1-173/(1981)

PASCAL FÖR PC-SYSTEM

STEN MINÖR
OSKAR PERMVALL

INSTITUTIONEN FÖR REGLERTEKNIK
LUNDS TEKNISKA HÖGSKOLA

SEPTEMBER 1981

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

Document name
MASTER THESIS

Date of issue
September 1981

Document number
CODEN:LUTFD2/(TFRT-5256)/1-171/(1981)

Author(s)
Sten Minör
Oskar Permvall

Supervisor
Hilding Elmqvist

Sponsoring organization
Swedish Board for Technical Development
(STU-80-3962)

Title and subtitle

Pascal for PC-systems
(Pascal för PC-system)

Abstract

Programmable controllers (PC) is today frequently used for logic control purposes, often integrated with computers. Usually the only language available for programming PC:s is rudimentary assemblers.

Using Pascal gives many advantages, as powerful data and program-flow structures, supporting the programmer to write structured and well-documented programs.

A code generator for a small PC (Satt-Electronics PBS-mini) is implemented. A Pascal compiler producing P-code has been used. Because of the limited hardware structure of the object PC, compared to a general computer, the P-code is partially evaluated. The code produced by the code generator has, after optimization, shown to be as efficient as hand-coded PC-code.

Key words

Classification system and/or index terms (if any)

Supplementary bibliographical information

ISSN and key title

ISBN

Language
Swedish

Number of pages
171

Security classification

Recipient's notes

DOKUMENTDATABLAD RT 3/81

Distribution: The report may be ordered from the Department of Automatic Control or borrowed through the University Library 2, Box 1010, S-221 03 Lund, Sweden, Telex: 33248 lubbis lund.

PASCAL
FÖR
PC-SYSTEM

Sten Minör
Oskar Permvall

Examensarbete
Handledare: Hilding Elmqvist

Institutionen för Reglerteknik
Tekniska Högskolan i Lund
Lund 1981

Vi skulle vilja tacka dem som hjälpt oss i vårt examensarbete. I första hand, institutionen för reglerteknik vid LTH och där speciellt vår handledare Hilding Elmqvist. Vi vill också tacka SATT-Electronics i Malmö för intresse och erhållet material. Vi tackar också institutionen för datateknik, DNA vid LTH för att vi fått nyttja deras dator och hjälp. Sist men inte minst vill vi tacka dem som står oss nära för att de på ett föredömligt sätt haft överseende med att vi alltför ofta varit upptagna med examensarbetet.

INNEHALLSFÖRTECKNING

0.	Inledning	3
1.	PC-SYSTEM	5
1.1	Allmänt om PC-system	5
1.2	Allmän beskrivning av PBS-mini	7
1.3	Centralenheten och input/output	10
1.4	PBS-mini:s instruktionsrepertoar	15
1.5	Programexempel	25
1.6	PBS-mini:s programmeringshjälpmedel	29
2.	PASCAL OCH P-KOD	31
2.1	Allmänt	31
2.2	Mellankoder	33
2.3	P-kompilatorn	35
3.	PASCAL FÖR PC-SYSTEM	41
3.1	Introduktion	41
3.2	Önskemål på språk för PC-system	43
3.3	Implementering av Pascal för PC-system	44
3.4	Pascal för PBS-mini	45
3.4.1	Datatyper	46
3.4.2	Tilldelningssatser och uttryck	47
3.4.3	If-satser	50
3.4.4	Repetitionssatser	52
3.4.5	Case	53
3.4.6	Procedures och functions	53
3.4.7	Write och read	57
3.4.8	Systemprocedurer	57
4.	BESKRIVNING AV KODGENERATORN	59
4.1	Översikt	59
4.2	Pass1 och pass2	59
4.3	Kodgenereringspasset	62
4.4	Stacken och behandling av uttryck	62
4.5	If-satser	68
4.6	Procedures och functions	70
4.7	Adressallokering för PBS-mini	75

5. OPTIMERING 77
5.1 Introduktion 77
5.2 Olika typer av optimering 78
5.3 Optimeringsprinciper i PBSOPT 80
5.4 Implementering av PBSOPT 87
5.5 Slutord om optimeringen 95

6. ERFARENHETER 97

7. PROGRAMEXEMPEL 99
7.1 Avesta jernverk 99
7.2 Gislaved däcklager 108
7.3 Sekvensstyrning 124
7.4 Skiftregister 127
7.5 Towers of Hanoi 130

Referenser med kommentarer 136

Appendix 1: Sammanställning av interpretering och kodgenerering av Pascal-element

Appendix 2: Programlistning av kodgeneratorn

Appendix 3: Programlistning av optimeringspasset

0. INLEDNING

PC-system (Programable Controllers) har under 70-talet mer och mer börjat användas inom industrin för digital styrning, dvs styrning där endast signaler av typen till-från är aktuella. PC-systemen var från början tänkta att ersätta vanliga "hoplödda" logik-system, och vann terräng på grund av möjligheterna att mjukvarumässigt bestämma och ändra ett systems funktion. En PC är i princip en enbits dator som, med ett relativt begränsat antal instruktioner, utför logiska beräkningar mycket snabbt.

Idag har emellertid PC-systemen blivit mer och mer sofistikerade och kapabla att utföra ganska avancerade operationer. Många olika typer av PC för olika typer av användning har dykt upp på marknaden, från små prisbilliga varianter för mindre applikationer, till stora industriella system ofta integrerade med datorer. Systemen har successivt förfinats att bli allt snabbare och innehålla allt fler hårdvarufaciliteter.

Mjukvarusidan av PC-systemen har dock varit dyster. Från tiden då systemen låg i sin vagga till i dag har inte mycket hänt. De flesta PC programmeras i ett enkelt språk på en nivå någonstans mellan assembler och maskinspråk. Detta har gjort att utvecklingskostnaderna för program blivit oanständigt stora i förhållande till de sjunkande kostnaderna för maskinvaran. Problemet är allmänt känt som "the software crisis".

Vårt arbete har varit att undersöka möjligheterna att använda programspråket Pascal för programmering av PC-system. En kodgenerator har implementerats, som genererar kod för ett av SATT-Electronics PC-system, PBS-mini. Översättningen från Pascal till PC-kod går över mellankod, s k P-kod. Denna P-kod genereras av en P-kods- kompilator framtagen vid Eidgenössige Technische Hochschule i Zurich, Schweiz. På grund av PBS-mini:s begränsade egenskaper jämfört med datorer Pascal är avsett för, partialevalueras P-koden och PBS-kod genereras. PBS-koden optimeras sedan för att uppnå högre effektivitet.

Pascal har visat sig vara ett kraftfullt programmeringshjälpmedel vid programmering av PC-system. Uttryck med symboliska namn på minnesceller och in-utgångar, data-strukturer såsom records och arrays, programflödesstrukturer såsom if-then-else, procedures och functions, ger möjligheter att skriva strukturerade och väldokumenterade program för PC-system, vilket bidrager till att hålla kostnaderna nere vid programutveckling. Effektiviteten hos den framtagna kodgeneratorn, m a p mängd genererad kod och minneskrav, har vid praktiska prov visat sig vara fullt jämförbar med kod skriven för hand i PC-kod.

1. PC-SYSTEM

1.1 Allmänt om PC-system

Ett PC-system, vad är det? Om det kan programmeras, varför inte kalla det för en dator, eller finns det något som skiljer det från en dator? Dessa frågor har ställts, när vi har berättat om vad vi gör i vårt projekt. Vi ska försöka besvara dem här.

Inom industrin har mottot ofta varit att förenkla handhavandet av och även snabba upp processerna. Att få så kontinuerlig drift som möjligt har oftast varit anledningen för satsningarna som gjorts på området. Att undvika avbrott i produktionen är viktigt, eftersom problemen, som uppstår kan vara mycket kostnadskrävande. Därför har automatisering av processer alltid varit intressant. Det började med Jaquards mekaniska vävstolar på 1700-talet. Utvecklingen har fortskridit med mer avancerad mekanik och har numera flyttats över på det elektriska området med strömmen som informationsbärare. På detta område var reläerna först och tillskansade sig en stor del av området processtyrning. När det i mitten på 1960-talet stod klart att mikroelektroniken hade kommit för att stanna, fick reläerna flytta ut, och mikroelektroniken tog hand om styrningen.

Första tiden gjordes styrenheter med specialdesignade kretskort för varje applikation. Detta medförde nykonstruktion av kretskorten för varje projekt, som skulle genomföras. Dessutom var service och underhåll svårt att genomföra, eftersom utvecklingen på mikroelektronikens område gick snabbt. Kretsar, som användes, var svåra att byta ut när de gick sönder, och de fanns kanske inte kvar på marknaden längre. Omkring 1970 uppenbarade sig de första PC-systemen som en del av lösningen på problemet. PC är en akronym för Programmable Controller. Lösningen innebär att all elektronik inom en typ av PC görs 'standardiserad'. Standardprodukter kan då användas för varje projekt. Framtagningstiden för ett projekt blev kortare, när styrningen av processen gjordes med programvara istället för hårdvara. Problemområdet flyttades alltså från kretskortskonstruktion till programmering med dess något annorlunda sätt att se på saken.

Varför är det då inte lämpligt att kalla PC-systemet för en dator? PC-systemet har något annorlunda egenskaper än en dator, och oftast har PC:n ordlängden 1 bit. Kanske kan PC:n kallas för enbits-dator. Till skillnad från datorn kan inte PC:n ha ett operativsystem. Den är också ganska låst i sin instruktionsrepertoar och mindre flexibel än datorn både före och efter att den knutits till en viss applikation. Efter det att PC:n har laddats med sitt program så är den låst till detta och exekverar det cykliskt i all evighet, om ström finns och inget oförutsett inträffar.

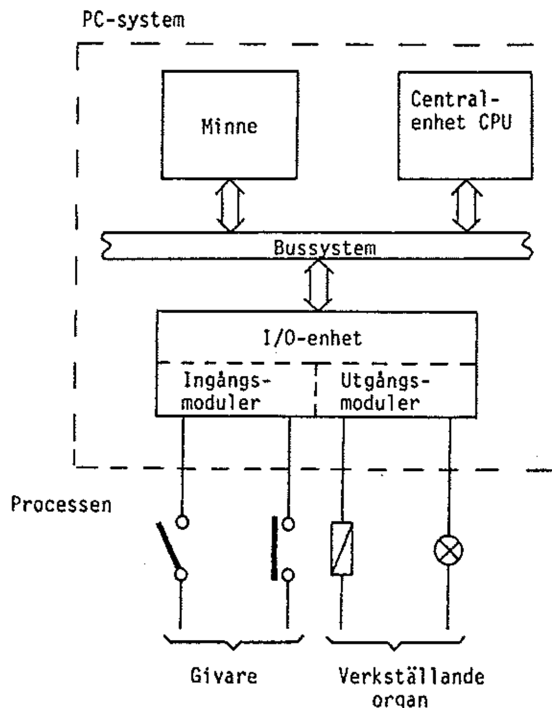
Istället för att likna PC:n vid en dator, bör den trots allt ses som ett sätt att specialdesigna system. Detta har många fördelar, t ex inga kallödningar, möjlighet till lättare felsökning och flexibel ändring i förhållande till verkliga integrerade kretsar. Det är också enkelt att reproducera redan konstruerade system, och att med lite ändringar anpassa det till en annan anläggning. Framför allt är man inte bunden till att anpassa processen till på marknaden befintliga kretsar utan kan göra programmet efter det mest logiska sättet att styra processen. PC-systemet har alltså tagits fram p.g.a. önskemål från processindustrin, där det finns många processer som kan styras binärt. PC-systemen har då fördelar framför datorn genom sin specialanpassning till den typen av processer. PC:n har inte behov av att vara generell i alla lägen. PC-systemen har slutligen flera andra fördelar, är lätta att installera, lätta att handha och är betydligt billigare än datorer. Också i prestanda, mätt i exekveringstid, är de bättre än datorer. De är även utformade att tåla tung industriell miljö med en svår elektrisk omgivning.

Resten av kapitlet kommer att ägnas åt beskrivning av ett speciellt PC-system och dess programmeringshjälpmedel: SATT-Electronics PBS-mini. Detta PC-system har använts som målmaskin för den kodgenerator vi har tagit fram. Den tekniska information, som lämnas i kapitlet, bygger dels på den erfarenhet en av oss har efter att ha arbetat med att programmera PBS-mini-system, dels på information ur manualer, artiklar och programexempel. Även genom diskussioner förda under konstruktionen av kodgeneratorn har vi fått ett bra grepp om PBS-mini:s inre funktion och arbetssätt. Det bör framhållas, att vi i resten av kapitlet endast diskuterar principer för hur ett PC-system kan se ut invändigt. Detta system kommer att ha stora likheter med PBS-mini, men vi har valt att betrakta det som ett system vilket som helst.

1.2 Allmän beskrivning av PBS-mini

PBS-mini i standardutförande består av en kortrack med plats för 19 kort. Av dessa kortplatser används 16 stycken för I/O-kort, och de övriga 3 för centralenhet, programminne och spänningsaggregat.

Det behöver inte vara 16 I/O-kort i racken för varje applikation. Antalet kort kan variera och gör därmed systemet relativt flexibelt. Till skillnad från andra system på marknaden tillåter PBS-mini godtycklig placering av de I/O-kort som ska ingå i systemet dvs alla 16 kan vara utgångskort eller alla vara ingångskort. Det finns många olika typer av I/O-kort beroende på vad som skall styras eller vilka signaler som skall plockas in från processen. Att det finns så många olika I/O-kort behöver inte programmeraren bekymra sig om då alla kort ser likadana ut ur programmets synvinkel.



figur 1.1
Översiktsbild på ett PC-system

De tre andra korten kan betraktas som fixa för systemet. Små variationer, som t.ex. olika minnestyper och storlekar, förekommer dock. Dessa tre kommunicerar via en buss som sitter längst in i racken. Denna buss hanterar överföringen av I/O-status mellan centralenhet och I/O-kort. Bussen har även hand om spänningsmatningen för denna kommunikation. Den talar också om för centralenheten på vilka platser i racken det finns kort, och av vilken typ de är (in- eller ut-kort).

Program-minneskortet har som mest plats för kapslar till 4k ord med ordlängden 16 bitar. Programminnet är försett med en batteribackup för att programmet inte ska försvinna vid strömavbrott, helgavstängningar m m. Program-minnet ska inte förväxlas med I/O-minnet, som innehåller aktuell status hos minnesceller, in- och ut-gångar. I/O-minnet har ordlängden 1 bit och rymmer totalt 1024 bitar. Centralenheten, som exekverar innehållet i programminnet, är uppbyggd med logiska kretsar och innehåller alltså ingen processor i vanlig mening. I/O-minnet, centralenheten och deras funktioner i samband med programexekvering och I/O-hantering beskrivs i ett särskilt delkapitel längre fram. Spänningsaggregatet försörjer racken med spänning, men det driver inte in- och ut-gångarna. Dessa drivs av en speciell spänningsmatning.

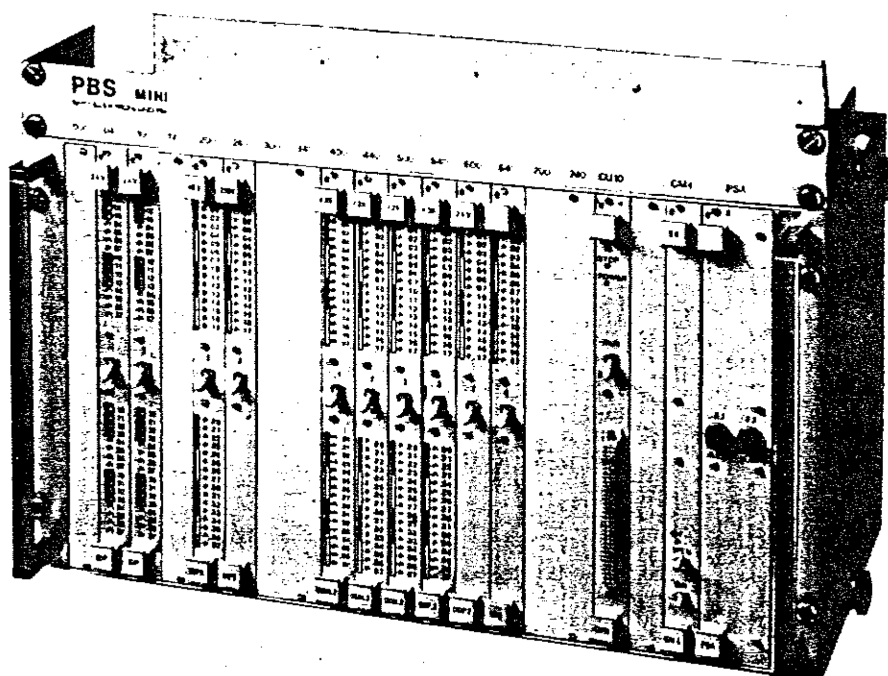
Sist i detta delkapitel ska vi räkna upp några fakta om och för PBS-mini:

512 in- och ut-gångar
(oktalt adresserade 0 - 777)

508 interna arbetsminnesceller
(oktalt adresserade 1004 - 1777)

De resterande fyra minnescellerna
är reserverade för centralenheten.

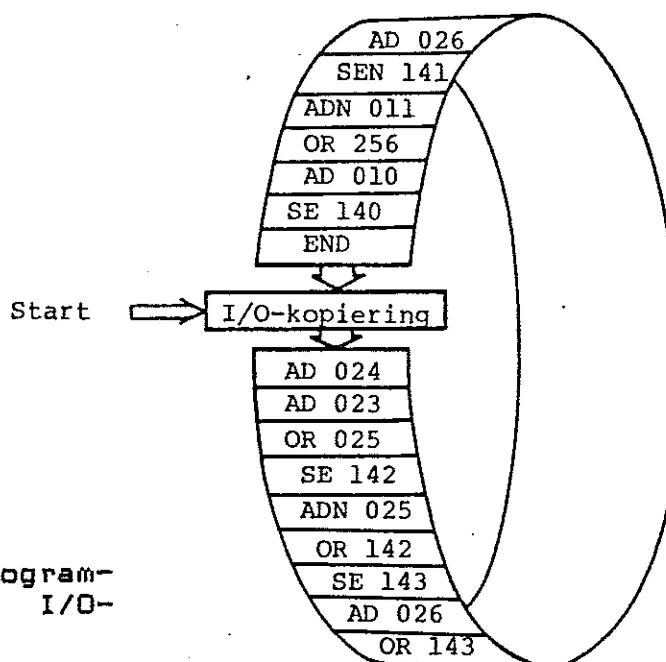
Den tillhandahåller två tidbaser för timers o.d., på
0.1 sek och 1 sek.
(De har de oktala adresserna 1000 och 1001)



figur 1.2
Bild på en PBS-mini i sin rack

1.3 Centralenheten och input/output

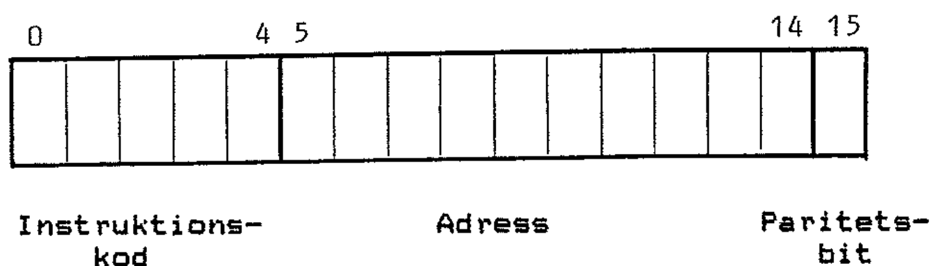
PBS-mini styrs av en centralenhet som exekverar innehållet i programminnet från dess början tills END nås eller programminnet tar slut. Alla operationer som utförs görs mot I/O-minnet. Programmet exekveras cykliskt, dvs "forever", men mellan varje exekvering utför centralenheten något som kallas för I/O-kopiering (se fig 1.3).



figur 1.3
Visar cyklisk programexekvering med I/O-kopiering.

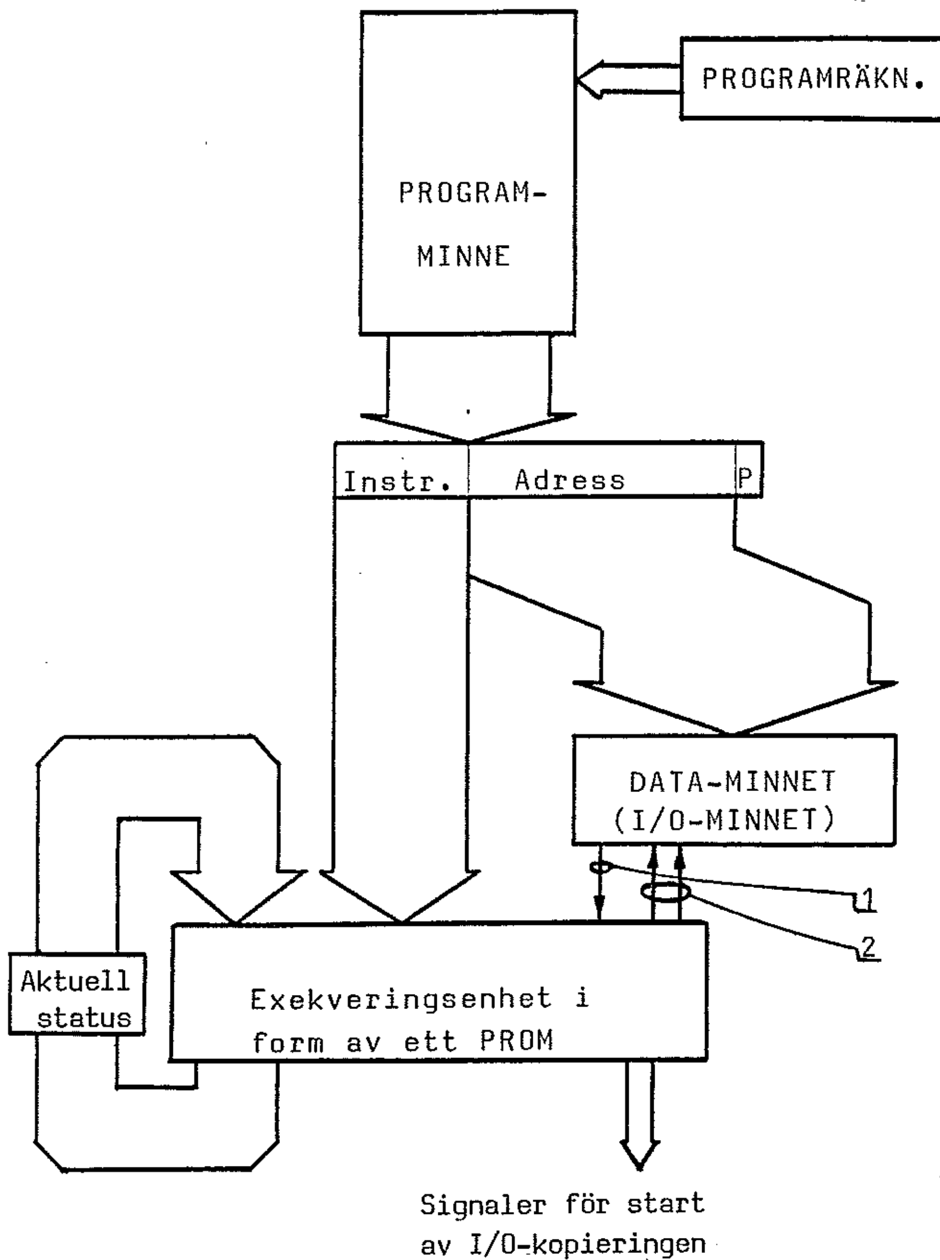
I/O-kopieringen innebär att I/O-minnet och in- och ut-gångarna utväxlar status. Det innebär att ingångarnas eventuella nya status skrivs in i I/O-minnet och utgångarnas status ändras ifall programmet ändrat deras status i I/O-minnet. Den här principen har flera fördelar. Bl a snabbar den upp programmet eftersom det endast behöver arbeta mot ett relativt snabbt halvledarminne istället för de fysiska ut- och in-gångarna. Mycket av I/O-kopieringen kan utföras parallellt, vilket medför kortare totaltid för en exekvering. Andra fördelar är att ingångars status ej ändras under programexekveringen, dvs transientfenomen på ingångarna har mycket liten möjlighet att störa körningen av programmet. Å andra sidan kan inte PBS-mini uppfatta processignaler som är kortare än totala cykeltiden mellan två I/O-kopieringar (~10-15 ms), men PBS-mini är inte avsedd för processer som har behov av så korta signaler.

Under programexekveringen beter sig centralenheten som en ren tillståndsmaskin med aktuellt tillstånd och aktuell programrad som indata. PBS-mini saknar alla möjligheter till hopp, så den kommer att tugga igenom programmet rad för rad från första till sista instruktion. För den fortsatta beskrivningen kan det vara intressant att se hur ett ord i programmet, dvs en programrad, är disponerat. Figur 1.4 visar ett ord.



figur 1.4
Figuren visar dispositionen av ett programminnesord.

Den principiella arkitekturen hos ett PC-systems centralenhet kan i bildform beskrivas som figur 1.5. Vi låter bilden tala för sig själv och hoppas att läsaren har nog elektronikkunskaper för att förstå kretslösningen. Observera att det finns en klockpulsgenerator för synkronisering av händelserna i centralenhetens kretsar. Den finns dock ej med i figuren.



Signalerna som har betecknats med siffror har följande funktion:

1. Nuvarande adresséns värde
2. Nytt värde till minnet och "Write enable"

figur 1.5
Figuren visar den principiella arkitekturen hos ett PC-systems centralenhet.

Av bilden på centralenhetens arkitektur framgår att det inte finns några explicita register förutom de två nödvändiga, instruktionsregistret och programräknaren. För att enkelt kunna tala om PC-systemet utan att gå in på detaljer, är det bra att kunna rubricera olika tillstånd hos centralenheten som att ett register har ett visst värde. P g a att datatypen i ett PC-system är "boolean" så behöver ett register endast kunna hålla två värden, dvs kan även liknas vid någon form av vippa.

Det kan för åskådlighetens skull vara lämpligt att betrakta olika tillstånd i maskinen som register. Fyra "register" är av intresse, nämligen:

- Arbetsregistret
- Parentesnivån
- Prioritetsregistret
- Skipvippan

Här nedan kommer vi kortfattat att förklara deras innebörd så som vi ser den.

Arbetsregistret :

Detta register håller aktuellt värde för den sats (förklaras i nästa delkapitel) centralenheten håller på och evaluera. I princip jobbar alla instruktioner mot arbetsregistret.

Parentesnivån :

PBS-mini har endast en parentesnivå. Här sparas all nödvändig information för att kunna fortsätta evalueringen av det totala uttrycket när högerparentesen nås, t ex vilken operator som skall verka mellan arbetsregistret och parentesuttrycket.

Prioritetsregistret :

PBS-mini har en del inbyggda prioriteter som kan jämföras med det som kallas "short evaluation" i vanliga programspråk. Ett exempel:

PBS-kod			Pascal-kod
AD	A		
AD	B		
OR	C	<=>	E:=A and B or C and D;
AD	D		
SE	E		

Om värdet av A and B är true så innebär det att E ska få värdet true oavsett värdet på C and D. Prioritetsregistret registrerar detta och ser till att arbetsregistret inte behåller fel värde efter evalueringen. Prioritetsregistret har även en speciell funktion i samband med timers.

Skipvippan :

Detta är den enda möjlighet som finns i PBS-mini att villkorligt exekvera instruktionssekvenser. Skipvippan sätts och återställs av speciella instruktioner. När den är satt så utförs inga instruktioner förrän den instruktion som återställer den nås. Då återupptas utförandet av instruktionerna igen. Märk att det inte är frågan om hopp utan centralenheten tuggar igenom instruktionerna som vanligt, dock utan att evaluera dem. Observera att skipvippan endast har en nivå, dvs det finns inga möjligheter till nästning.

I de följande delkapitlen kommer vi att referera till funktion och innehåll hos de olika registren. Speciellt i samband med beskrivningen av instruktionsrepertoaren.

1.4 PBS-mini:s instruktionsrepertoar

PBS-mini programmeras med ett antal enkla logiska instruktioner av typen AND, OR m fl . Den har också ett antal specialinstruktioner för timers, flanktriggning m m . Det finns 25 olika instruktioner (se fig 1.6). Det finns två typer av instruktionsformat:

1: <instruktionskod>

2: <instruktionskod> <adress>

En indelning av instruktionerna efter vilket format de har kan göras på följande sätt:

format 1:
AP, OP, RP, SS, SSN, RS, CP, NPW, NOP, END

format 2:
AD, ADN, OR, ORN, EX, EXN, SE, SEN, SSR, RSR, EDGE, CNT+, CNT-, COT+, COT-

Med <adress> avses en minnescell, ingång eller utgång. På denna skall instruktionen verka. P g a den tidigare nämnda principen med I/O-kopiering så verkar alla instruktioner mot I/O-minnet, även benämnt dataminnet. Varje instruktion tar upp en rad i programmet, oavsett vilket format den har. En programrad motsvarar ett ord i programminnet, dvs eftersom programminnet är max 4k ord så kan programmet vara max 4096 rader långt.

PBS-mini kan endast hantera data av typen ettor och nollor, och endast en etta eller nolla i taget. Det motsvaras av typen boolean i de flesta programspråk. Det innebär att instruktionerna kan konstrueras för att enbart arbeta på den datatypen, vilket också innebär att ingen hänsyn behöver tas till andra datatyper och dess representation. Ävenledes är alla instruktioner för hantering av olika adresseringsmoder onödiga.

Instr. nr	+		-	
0	RP	)	NOP	NO OPERATION
1	AD	AND	ADN	AND NOT
2	OR	OR	ORN	OR NOT
3	SE	SET	SEN	SET NOT
4	SS	SET SKIP	SSN	SET SKIP NOT
5	EDGE	EDGE	RS	RESET SKIP
6	CNT+	COUNT (PRESET=1)	CNT-	COUNT (PRESET=0)
7	COT+	COUNT OUT = 1	COT-	COUNT OUT = 0
8	OP	+(	AP	*(
9	END	END	CP	CLEAR PBS
10 (A)	SSR	SET SR-FLIPFLOP	RSR	RESET SR-FLIPFLOP
11 (B)	EX	EXCL OR	EXN	EXCL OR NOT
15 (F)	NPW	NOT PROGRAMMED WORD		

figur 1.6
PBS-mini:s instruktionstabell

Det går inte att föra in värdet ett eller noll direkt i programflödet utan det måste göras via en adress till dataminnet, eller via ett av de fyra s k registren (arbetsregistret, parentesnivån, prioritetsregistret och skipvippan). Detta framgår tydligt av de två instruktionsformaten, då där inte finns något format som direkt hanterar data utan endast via adresser till dataminnet. Registren kan man inte adressera direkt utan de nås implicit eller explicit via vissa instruktioner, se nedan.

Då datatypen har en så enkel utformning och man i möjligaste mån vill underlätta arbetet för den som ska programmera en PBS-mini, så har de flesta instruktionerna utformats med två moder. Dessa moder representeras med ett prefix eller suffix, alltid bestående av ett tecken, till den gemensamma grundkoden. Nedan visar vi vilka olika moder som finns och vilka instruktioner som har den moden. Det framgår också vilka moder som är varandras komplement.

' ' och 'N'	SE	SEN
<suffix>	AD	ADN
	OR	ORN
	EX	EXN
	SS	SSN
'+' och '-'	CNT+	CNT-
<suffix>	COT+	COT-
'S' och 'R'	SSR	RSR
<prefix>		

Det går att ge de olika moderna namn efter deras funktion. Den första i figuren skulle då bli icke-inverterande resp inverterande, medan i de båda andra så associeras 'S' och '+' till värdet ett resp 'R' och '-' till värdet noll. Allt detta hänger samman med deras funktion som beskrivs längre ner. De instruktioner som har två olika moder har också oftast instruktionsformatet nr 2, dvs moden avser att verka på instruktionens adressdel. Det är bara en instruktion som avviker från mönstret 'SS' resp 'SSN'. Här är moden avsedd att verka på arbetsregistret och skipvippan istället för någon cell i dataminnet.

Ett flertal av instruktionerna kan föregås av ett villkor som byggs upp med andra instruktioner. Det går då att dela in PBS-mini-programmet i en form av satser enligt följande BNF-syntax:

<villkor> ::= <villkorsinstr> <villkor> | <villkorsinstr>

<sats> ::= <villkorslösinstr> | <villkor> <satsslutinstr>

| <satsslutinstr>

De odefinierade symbolerna bestäms av följande mängder av instruktionskoder:

<villkorsinstr> tillhör mängden
{AD,ADN,OR,ORN,EX,EXN,AP,OP,RP,EDGE}

<villkorslösinstr> tillhör mängden
{RS,END}

<satsslutinstr> tillhör mängden
{SE,SEN,SS,SSN,SSR,RSR,CP}

I denna definition av en sats så har vi inte splittrat upp den ytterligare med avseende på formatet utan det får tänkas följa med instruktionskoden. Definitionen täcker endast de s k vanliga satserna, special och tomma instruktioner finns inte med utan behandlas enbart nedan.

Villkoret i en sats evalueras instruktion för instruktion. Enda sättet att föra över information mellan olika ställen i programmet är via registren. Då kan det vara väsentligt att veta från vilka register en instruktion hämtar sin information och vilka register den påverkar. Figurerna som kompletterar instruktionsbeskrivningarna nedan, är till just för att ge en överblick på informationsflödet mellan registren vid olika instruktioners evaluering. Det finns dock en del implicita händelser hos vissa instruktioner som inte är med i figurerna, bl a gäller det satsslutinstruktionerna. De ställer alltid arbetsregistret i sitt utgångsläge och även t ex parentesnivå och prioritetsregister ställs i sina utgångslägen.

Det går att diskutera om den principen som används för evalueringen, dvs arbetsregister och prioritetsregister är att föredra framför att enbart ha arbetsregister som datorer normalt har. Nu är det så i detta fallet att registren har fler funktioner än enbart villkorsevalueringen t ex i samband med timers, vilket ger motiv att ha det på det sättet.

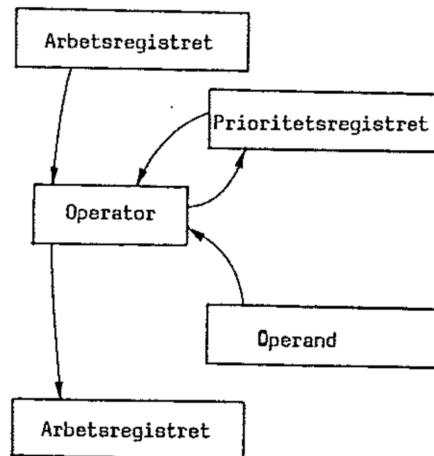
Vi kommer här att beskriva varje instruktion och dess funktion. De kommer att grupperas efter ovanstående mängder.

<villkorsinstr> :

AD <adress> , ADN <adress>

OR <adress> , ORN <adress>

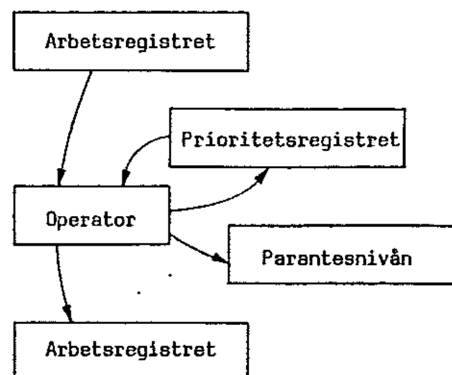
EX <adress> , EXN <adress>



Utför and resp and not, or resp or not, exor resp exor not mellan värdet hos arbetsregistret och värdet på <adress>. Märk att inverteringen verkar på värdet hos adressen. Instruktionerna påverkar även prioritetsregistret.

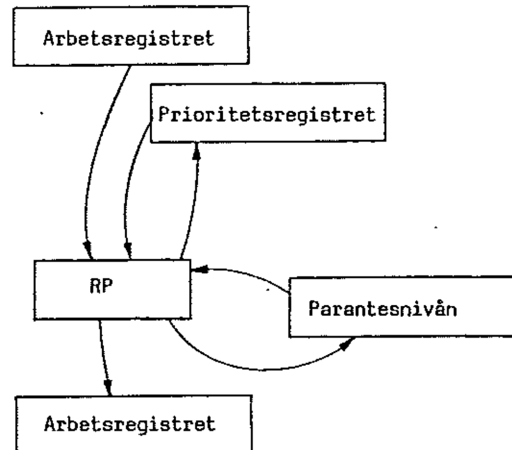
AP

OP



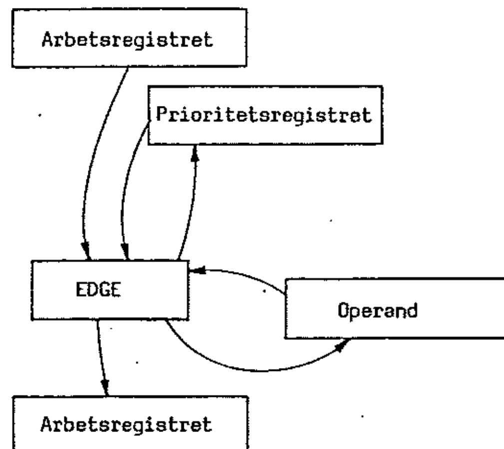
Utför and resp or mellan arbetsregistret och det uttryck som finns mellan parenteserna. Här sparas alltså värdet hos arbetsregistret undan i parentesnivån och arbetsregistret ställs i sitt utgångsläge för att kunna evaluera uttrycket mellan parenteserna. Även prioritetsregistret påverkas. Endast en parentesnivå är tillåten.

RP

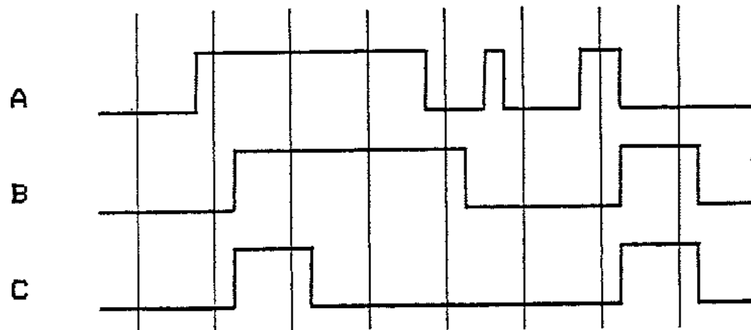


Avslutar ett parentesuttryck och bildar det totala resultatet av parentesuttrycket och det värde resp operator som föregick det. Parentesnivån ställs i utgångsläge och prioritetsregistret ställs i lämpligt läge.

EDGE <adress>



Används för att detektera tillståndförändringar s k flanker. Instruktionen detekterar endast positiv flank hos det uttryck som föregår den. Dvs när värdet hos uttrycket går från 0 till 1, instruktionen ger värdet 1 ut som resultat då flanken uppstår och behåller detta värde tills instruktionen passeras nästa gång (se fig på nästa sida).



Denna kodsekvens ger det tidsschema som visas ovan, då en signal uppträder på A:

```
AD   A
EDGE B
SE   C
```

(Obs! De lodräta strecken i figuren motsvarar I/O-kopieringar)

Märk att pulser som är kortare än tiden mellan två I/O-kopieringar och om pulsen hamnar mellan två sådana, kommer den ej att detekteras av PBS-mini:n (jmfr sampling av signaler i andra sammanhang). Däremot om den inträffar "över" en I/O-kopiering så kommer en puls att uppstå i PBS-mini:n (se figuren).

<villkorslösinstr> :

RS

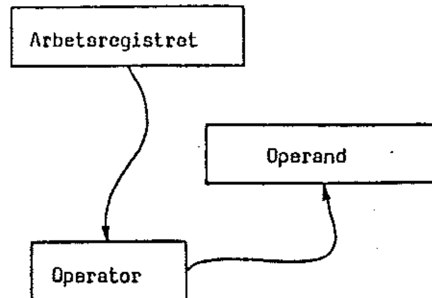
Återställer skipvippan till utgångsläget. Att den är villkorslös beror på att om skipvippan vore satt, skulle villkoret inte gå att evaluera. Instruktionen avslutar alltså ovillkorligt en villkorlig sektion i PBS-programmet.

END

Denna instruktion avslutar programmet och påbörjar I/O-kopieringen. Den kan således inte ha något villkor.

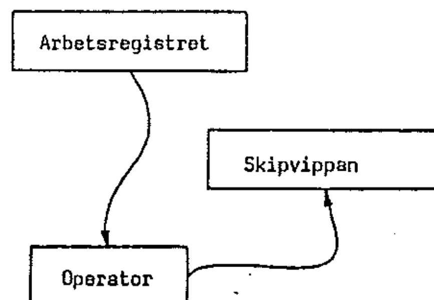
<satsslutinstr> :

SE <adress> , SEN <adress>



Överför arbetsregistrets innehåll till <adress> i minnet. Arbetsregistret innehåller här det evaluerade villkorets värde.

SS , SSN

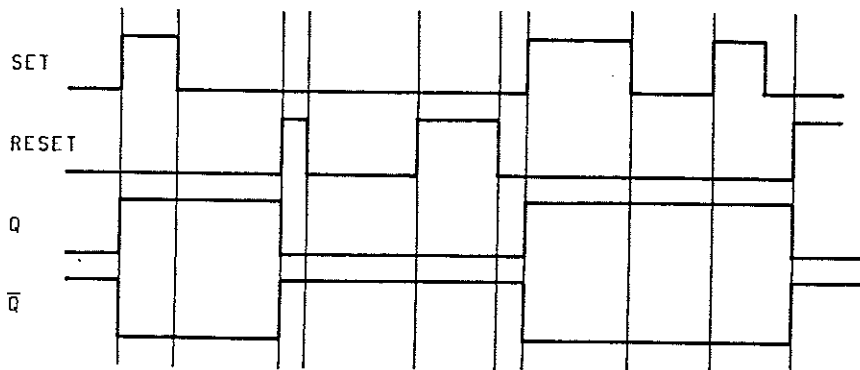


Dessa instruktioner används för att villkorligt blockera exekveringen av instruktionerna fram till nästa RS. Det är skipvippan som håller reda på om centralenheten ska exekvera en instruktion eller ej. Den sätts alltså av dessa instruktioner.

SSR <adress> , RSR <adress>

Se figuren under SE och SEN.

Vid användning av dessa instruktioner på en minnesadress kommer den adressen att fungera som en SR-vippa. Det blir dock inte fullständigt ekvivalent med en SR-vippa eftersom det inte går att påverka både S- och R-ingångarna samtidigt. De är dock funktionellt ekvivalenta (se figur nedan).



Figuren visar ett tidsschema för en SR-vippa. Används andra instruktioner på adressen så kan det hända att SR-funktionen upphävs.

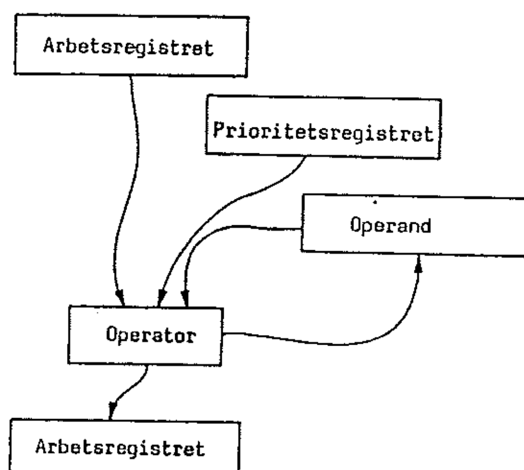
CP

Denna satsslutinstruktion är lite speciell, ty den nollställer hela I/O-ram:et om instruktionens villkor är sant. I/O-ram:et förblir nollställt tills villkoret blir falskt igen.

<speciella instruktioner> :

CNT+ <adress> , CNT- <adress>

COT+ <adress> , COT- <adress>



CNT +/- är timerinstruktioner för den räknande delen av timern. '+' och '-' talar om vilket värde adress ska initieras till när timern får initieringssignal. Som beskrevs ovan så är '+' ⇔ 1 och '-' ⇔ 0.

COT +/- avslutar en timer och ger sin adress ett värde beroende på om instruktionen har ett plus eller minus. Här är '+' ⇔ 0→1 och '-' ⇔ 1→0 då timern går ut.

De här instruktionerna kan även användas för att implementera andra specialfunktioner än timers t ex räknare m m. För information om detta hänvisar vi till SATT-Electronics manualer för PBS-mini.

NPW , NOP

Slutligen finns två instruktioner som endast är till för utfyllnad och för att underlätta vid ändringar. De påverkar inte tillståndet i centralenheten på något sätt, utan passerar endast rakt igenom.

1.5 Programexempel

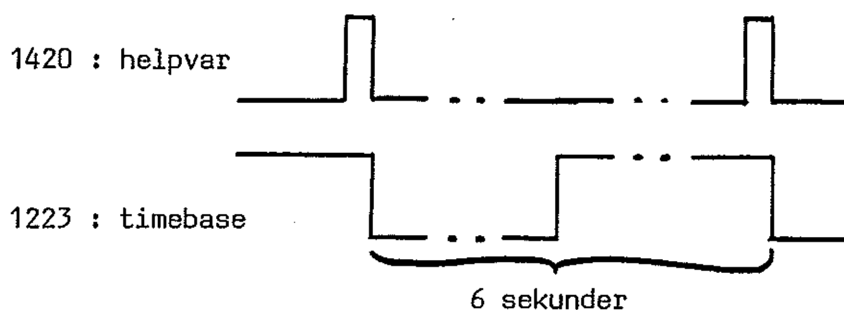
Här ska visas några exempel på delar av PBS-mini-program. Vi kommer också att visa motsvarande uttryck i Pascal, för att knyta an till följande kapitel. Vi visar endast mer speciella uttryck, eftersom vi antar att läsaren är tillräckligt bevandrad i konstruktionen av vanliga logiska uttryck.

Exempel 1 :

En egen-definierad tidbas bestående av två timers som driver varandra. Tidbasen kommer att ha periodtiden 6 sekunder. Se även tidsschemat under PBS-koden för förståelse av funktionen.

PBS-kod :

```
AD   1420          ADN  1223
OR   1001          OR   1001
EDGE 1421          EDGE 1224
CNT+ 1422          CNT+ 1225
CNT+ 1423          CNT+ 1226
COT+ 1223          COT+ 1420
```



Pascal-kod :

```
timer(not helpvar,false,puls1hz,timebase,3);
timer(timebase,false,puls1hz,helpvar,3);
(* 3:orna är tillslagstiden för resp. timer *)
```

Exempel 2 :

Den tidbasen vi konstruerade i förra exemplet går sedan att använda i andra timers. De kommer då att räkna med noggranheten 6 sekunder. Den här timern är konstruerad för att räkna till 1 minut. Motsvarande timer med 1-sekunds tidbasen hade innehållt 6 st CNT-instruktioner.

PBS-kod :

```
ADN    15
OR   1223
EDGE 1227
CNT- 1230
CNT+ 1231
CNT- 1232
CNT+ 1233
COT+   43
```

Pascal-kod :

```
timer(inexpr,false,timebase,output,10);
```

Exempel 3 :

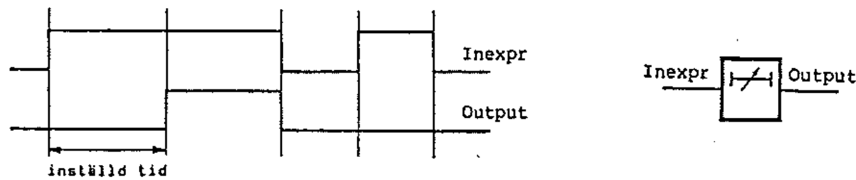
Vi vill ge exempel på ännu en timer driven av tidbasen i exempel 1. Den här kan stoppas tillfälligt, om variabeln holdexpr (26) är sann. Inställda tiden är 36 sekunder. Efter exemplet visar vi två tidsscheman och hur man modellerar respektive timer i logikscheman.

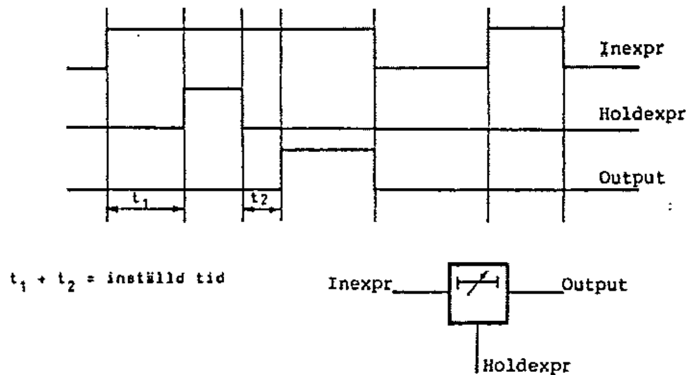
PBS-kod :

```
ADN    15
OR   1223
EDGE 1234
ADN    26
CNT- 1235
CNT+ 1236
CNT+ 1237
COT+   43
```

Pascal-kod :

```
timer(inexpr,holdexpr,timebase,output,6);
```





Exempel 4 :

Vippor finns det användning för i PC-program. Vi har tänkt visa en D-vippa eller som den allmänt kallas : fördröjningselement. Den vippan vi beskriver kan även explicit sättas och återställas. Återställningen är dominant. Vi har också valt att använda en speciell variabel för klockpulsen för att kunna använda den på andra ställen i ett PC-program.

PBS-kod :

```
AD     20    ; clock
EDGE 1240    ; puls
SE   1241
AD   1241
SSN
AD     21     ; D data in
SE    200     ; D data ut
RS
AD     23     ; set
SSR   200
AD     22     ; reset
RSR   200
```

Pascal-kod :

```
puls:= edge(clock,helpedge);
if puls then Ddataut:= Ddatain;
if set then Ddataut:= true;
if reset then Ddataut:= false;
```

Exempel 5 :

I förra exemplet blev kanske Pascal-koden ovanligt lång, men å andra sidan lättare att förstå. Skall ett sådant element som t ex vippor användas på fler ställen är det lämpligt att göra en procedure av det.

Som sista exempel vill vi visa en vippa som i sitt grundutförande saknar motsvarighet i vår Pascal-notation, sådan den är idag. Det är en T-vippa, dvs en vippa som skiftar mellan två tillstånd för varje klockpuls. Den kan också sättas direkt via en separat ingång.

PBS-kod :

```
AD      11
OR    1241
CNT+   100
SE    1242
```

Det går dock att behålla funktionen, men formulera den på annat sätt så att den har en motsvarighet i Pascal-notationen. Det kan göras på följande sätt:

PBS-kod :

```
AD    1241
SSN
AD      11
ORN    100
SE     100
RS
```

Pascal-kod :

<u>if</u> puls <u>then</u> Tdataut:= set <u>or</u> <u>not</u> Tdataut;

Med exempel 5 vill vi visa att det finns alltid program skrivna i assembler som inte har någon motsvarighet i ett högnivåspråk. Därmed är det inte sagt att att assembler skulle vara ett bättre språk. Högnivåspråkets huvudsakliga syfte är att förenkla för programmeraren, och öka läsbarheten.

1.6 PBS-minis programmeringshjälpmedel

Det finns ett antal olika programmeringshjälpmedel till PBS-mini. En del är baserade på en dator som man kan ansluta en terminal till och på det sättet kommunicera med PBS-mini:n. Andra hjälpmedel bygger på principen att direkt läsa och skriva i programminnet hos PBS-mini, dvs endast behandla instruktioner en i taget.

Vi kommer i detta kapitlet endast att intressera oss för de datorbaserade programmeringshjälpmedlen. Vi känner till tre olika :

- Vanlig editering av PBS-kod
- Editering med hjälp av reläsymboler
- Editering med ABC-80

Alla tre använder endast vanlig PBS-kod mer eller mindre finessrikt. Det finns såvitt vi vet ingen motsvarighet till det projekt som vi gjort, dvs högnivåspråk till PC-system. De tre programmeringshjälpmedlen beskrivs kortfattat nedan.

Gemensamt för alla tre programmeringshjälpmedlen är att de måste alla anslutas till PBS-mini:n via en enhet som kallas PROG-11. Denna innehåller en mikrodator och programvara i PROM för att kunna hantera PBS-mini:s programkod.

Vanlig editor :

Här anslutes någon form av terminal till PROG11:an på den PBS-mini som skall programmeras. Editorn är uppbyggd efter samma principer som en vanlig datoreditor, dock är den anpassad till PBS-mini på så sätt att det endast går att mata in korrekta instruktioner för PBS-minin. Editorn har kommandon för Insert, Erase, List m m, plus att den har kommandon för lagring av PBS-program på sekundärminne i form av kassettband.

Reläsymbolsprogrammering :

För att kunna använda detta hjälpmedel måste man ha en speciell bildskärm med knappar för reläsymbolerna. Det här sättet att programmera PBS-mini kan jämföras med någon form av skärmorienterad editor. Även här finns kommandon för Insert, m m, plus kommandon för visning av hela reläuttryck på skärmen. Det går också att automatiskt konvertera mellan PBS-kod och reläsymboler.

Editering med ABC-80 :

För att kunna editera PBS-program på det här sättet måste man ha tillgång till två datorer, dels PROG11:ans mikrodator och dels en ABC-80 med flexskivor. Det finns flera fördelar med den här metoden jämfört med de andra två. En av vinsterna är att programmen kan förvaras på flexskiva, det medför snabbare åtkomst än med kassettbandspelare. Det finns också en mängd andra möjligheter t ex namn på in- och ut-gångar, kommenterar i programlistan m m. Dock programmerar man fortfarande i ren PBS-kod.

2. PASCAL OCH P-KOD

2.1 Allmänt

Pascal utvecklades ursprungligen av Niklaus Wirth på Eidgenössische Technische Hochschule (ETH) i Zurich. Den första versionen av språket kom 1968, och var då i huvudsak avsedd att användas för undervisning i programmering. Populariteten har därefter ökat, och Pascal används idag som ett allmänt programmeringsspråk, även kommersiellt i allt större utsträckning.

Pascal har många egenskaper som brukar gå under beteckningen strukturerad programmering, som har varit ett av honnörsorden under 70-talets utveckling på mjukvarusidan. Ett program ska vara lätt att läsa och förstå, och den logiska strukturen ska vara lätt att genomskåda. Språket ska också kännas naturligt att använda.

Säkerheten är en viktig aspekt på moderna programmeringsspråk. Genom att begränsa programmerarens frihet och införa strikta krav på hur ett korrekt program ska se ut, kan man i många fall detektera fel redan under kompileringen. Att hitta så många fel som möjligt på ett tidigt stadium är fördelaktigt. Kan felaktiga eller oavsiktliga konstruktioner upptäckas redan under kompileringen spar man i regel mycket tid, då man slipper svårhittade run-time fel.

Pascal är ett relativt litet språk, med ett litet antal generella språkelement, vilket gör att man med ett icke oöverstigligt mått av energi kan lära sig hela språket. Utifrån dessa grundläggande begrepp kan sedan mer komplexa strukturer konstrueras av användaren allt efter önskemål. Språket är blockorienterat med en struktur som liknar Algol och Simula. Trots begränsad storlek omfattar Pascal många kraftfulla begrepp såsom ett välutvecklat typbegrepp, pekarstrukturer, dynamisk minnesallokering och rekursivitet. Satser som if-then-else, while-do, repeat-until och case-of, tillsammans med uppdelningen av programmmet i procedurer, befrämjar i hög grad god programstruktur.

Pascal får nog räknas till klassen "generella" programmeringsspråk, för användning framförallt inom området tekniska och numeriska beräkningar. På grund av sin generalitet kan det i viss mån användas även för exempelvis simulering och formelbehandling. För realtidsapplikationer har vidareutvecklingar av Pascal gjorts, som t ex Concurrent Pascal, vilket tillför begrepp som parallella processer och abstrakta datatyper.

När Pascal utvecklades fanns också krav på att det skulle vara lätt att ta fram kompilatorer. Språket är i många avseenden konstruerat att vara "kompilatorvänligt", vilket gör det lättare att ta fram en kompilator för Pascal än för många andra språk. En av anledningarna till att Pascal är så spritt som idag, torde vara att den portabla kompilator som togs fram av U. Amman vid ETH, Zurich, 1973. Kompilatorn är skriven i Pascal och producerar en mellankod, s k P-kod. P-koden, som ligger på assemblernivå, är kod för en tänkt dator. För att kunna exekvera ett program i P-kod måste det alltså översättas till kod för målmaskinen av en separat kodgenerator, unik för varje datortyp. Ett annat sätt är att direkt interpretera P-koden. Metoden är enklare att implementera, men är ur effektivitetssynpunkt klart underlägsen.

Det finns dock en rad invändningar mot Pascal. Standardiseringen av språket är idag mycket vag. Varje implementering har sina egna hemmasnickrade "smart"-lösningar på önskade konstruktioner, vilket i hög grad begränsar portabiliteten på program skrivna i Pascal. Dålig säkerhet vid användning av pekare, rudimentär (usel) stränghantering, svårigheter vid användning av fält som parametrar till procedurer och ett mindre antal andra olägenheter brukar framföras till Pascals nackdel. Eftersom dessa invändningar gäller element utanför det subset vi använder för PC-programmering kan de lämnas därhän i detta sammanhang.

2.2 Mellankoder

Metoden att kompilera ett språk till en mellankod, för att därefter generera kod utifrån den, har blivit allt vanligare. På så sätt fås ett naturligt snitt mellan kompilering och kodgenerering för objektdatorn. I första passet omhändertas lexikalisk analys, "parsing", felhantering och naturligtvis generering av mellankoden. Denna bit av kompilatorn kan skrivas i samma språk som kompilatorn är skriven för och blir lätt att flytta mellan olika datortyper. Det finns även exempel på automatiska kompilatorgeneratorer som, från en given grammatik för ett språk, genererar huvuddelen av detta pass.

Kodgenereringen sker därefter i ett separat pass. Här finns inga generella metoder för implementering, utan varje mellankodsinstruktion måste översättas till den (de) instruktioner den motsvarar för just den dator implementeringen avser. Detta pass kan inte bli portabelt, utan är unikt för varje datortyp och måste således skrivas för hand då en kompilator ska flyttas från en datortyp till en annan. Arbetet att skriva kodgenereringspasset är dock litet i jämförelse med den arbetsinsats som krävs för att skriva en hel kompilator. Att flytta en kompilator på detta sätt kallas ofta "bootstrapping" (att lyfta sig själv i pjäxsnörena).

Mellankoden ska vara så generell som möjligt, d v s inte ta hänsyn till en viss maskins hårdvarukonfiguration. Det medför att koden måste ligga på en ganska hög nivå, t ex addera två heltal. Hur additionen ska gå till behöver inte framgå, utan implementeras på effektivast sätt för avsedd maskin vid kodgenereringen. Man behöver då inte ta hänsyn till om målmaskinen är en stackorienterad maskin, om instruktionsrepertoaren tillåter additioner direkt till minnet e t c. Andra skäl till att mellankoden bör ligga på en hög nivå är att många datorer har specialinstruktioner för att utföra en viss operation. Skulle mellankoden då vara på en lägre nivå, skulle det betyda att en hel sekvens mellankod skulle behöva kännas igen av kodgeneratorn och därefter bytas ut mot respektive specialinstruktion, vilket kan vara besvärligt och kräva en avsevärd look-ahead.

Att eliminera behovet av look-ahead är också en önskvärd egenskap hos mellankoder. Det ska vara möjligt att generera kod genom att bara titta på en rad åt gången, och sekventiellt gå igenom mellankoden vid kodgenerering. Varje instruktion ska alltså innehålla all information som behövs för att generera kod. Man ska inte behöva hålla reda på något från föregående instruktioner, om man för tillfället är inuti en if-sats exempelvis.

Ytterligare en fördel med att generera kod från en mellankod är att koden förutsättes alltid vara syntaktiskt riktig. Man behöver således inte spilla någon kraft på felhantering.

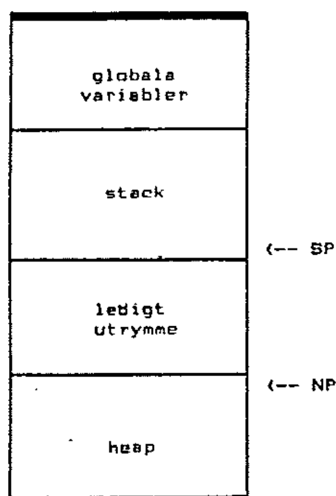
2.3 P-kompilatorn

P-kompilatorn vi använder är utvecklad på ETH, Zurich. Den genererar P-kod för en hypotetisk stackorienterad dator. P-koden är en hitta-på-assembler med instruktioner som motsvarar de instruktioner som behövs för att köra Pascal, och som brukar finnas i en eller annan form på en vanlig dator.

Stack-datorn består av 4 register och ett minne. Registerna är:

1. PC Program Counter
2. SP Stack Pointer
3. MP Mark Pointer
4. NP New Pointer

Minnet i stack-datorn är delat i två delar: programminne och dataminne. Programminnet innehåller instruktionerna som ska exekveras. PC blir således en pekare till programminnet.



Dataminnet är organiserat i två delar. Den ena, som kallas heap (hög), är till för data som allokeras dynamiskt under körningen med standard-proceduren new( ). Den andra delen, stacken, är till för dels variabler, dels beräkning av uttryck. Längst ner i stacken (längst upp i figuren) ligger alla globala variabler. Varje gång en procedur anropas kommer en ny data-area allokeras i stacken för procedurens lokala data.

SP, stackpointern, markerar stackens topp. MP, markpointern, pekar på det ställe i stacken där senaste anropet skedde, d v s där data-arean till den procedur som för närvarande exekveras börjar. All evaluering av uttryck sker längst upp på stacken.

Evalueringen sker, som är naturligt för stackstrukturen, i RPN (Reverse Polish Notation). Instruktioner som "load" laddar upp ett värde i stacken. "Store" lagrar på samma sätt ett värde som tas överst i stacken i angiven minnescell. Operationer som "add" e t c kommer ta stackens två översta element, addera dem, och lägga resultatet överst på stacken. Stacken kommer sålunda växa och sjunka succesivt med instruktionerna. "Load" gör implicit push på stacken medan exempelvis "add" implicit gör pop.

Anrop av procedures och functions är lite speciellt. Ett anrop kan se ut som följer:

```
MST          0  ;  Mark stack
LDOB        36  ;  Load boolean at address 36 in stack
LDOB        38  ;  Load boolean at address 38 in stack
CUP      0  L9  ;  Call user procedure at label L9
```

När en ny procedure (function) anropas ska ett nytt minnesblock allokeras för lokala data till proceduren. Att ett nytt block data skapas vid varje anrop är en förutsättning för rekursiva procedurer. Instruktionen MST lägger först upp ett mark-stack-block (fig 2.2). Blocket innehåller 4 minnesceller. Den första är till för undanlagring av returvärde (endast aktuellt vid function). De två följande innehåller blockets statiska resp dynamiska fader, d v s i vilket block proceduren är deklarerad resp var den är anropad. Den fjärde minnescellen innehåller återhoppsadressen, där exekveringen ska återupptas då proceduren är klar. MP sätts att peka på mark-stack-blocket. Därefter laddas procedurens aktuella parametrar upp i stacken med två LDOB (två parametrar). CUP överlämnar kontrollen till proceduren, som exekverar tills ett RETP, return from procedure, påträffas. Procedurens minnesblock frigörs då och exekveringen återupptas vid instruktionen efter anropet.

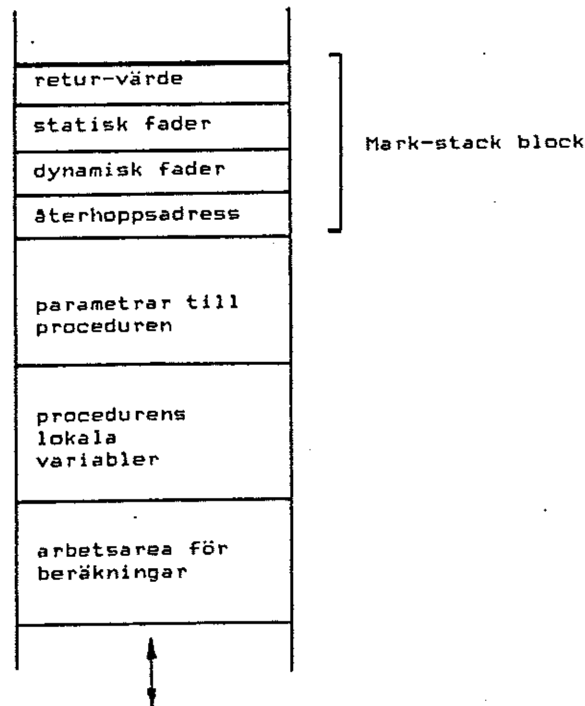


fig 2.2 Stackens utseende vid proceduranrop

En P-kods instruktion har formatet:

OPERATION<typ> <P> <Q>

Operationen består av en mnemonisk kod på tre bokstäver, eventuellt direkt följd av typen på operander eller resultat. P och Q är operander som anger värden eller adresser beroende på instruktionen.

För att underlätta läsandet av exempel, speciellt kap 4, följer en lista på samtliga P-kods-instruktioner med en kort kommentar till varje. Asterisk i tabellen markerar om P- respektive Q-fält ingår i instruktionen samt om instruktionen har en typbokstav efter de tre bokstäver som utgör själva instruktionen. Typbokstaven anger vilken typ av data instruktionen opererar på. Typbokstäverna är B-boolean, I-integer, R-real, A-address, S-set, N-nil, P-procedure och C-character.

Instruktion	Typ	P	Q	Kommentar
ABI				produce absolute value of integer
ABR				produce absolute value of real
ADI				produce sum of integer
ADR				produce sum of reals
AND				perform boolean and
CHK	*	*	*	check that stack top)=P and (=Q
CHR				convert integer to character
CSP			*	call standard procedure
CUP		*	*	call user procedure
DEC	*		*	decrement stack top by amount Q
DIF				evaluate set difference
DVI				integer divide
DVR				real division
ENT		*	*	enter block
EOF				test for end of file condition
EQU	*			test for equality
FJP			*	jump to Q if stack top false
FLO				float the element below stack top
FLT				float the top of stack
GEQ	*			test for greater than or equal
GRT	*			test for greater than
INC	*		*	increase stack top by amount Q
IND	*		*	indexed fetch
INN				test for membership (in)
INT				set intersection
IOR				perform boolean inclusive or
IXA			*	compute indexed address
LAO			*	load base level address
LCA			*	load address of constant in Q
LDA		*	*	load address
LDC	*		*	load constant given in Q-field
LDO	*		*	load contents of base level address
LEQ	*			test for less than or equal to
LES	*			test for less than
LOD	*	*	*	load contents of address
MOD				modulus (on contents of stack top)
MOV			*	move data block
MPI				multiply integers
MPR				multiply reals
MST		*		mark stack
NEQ	*			test for not equal
NGI				negate integer
NGR				negate real
NOT				perform boolean not

Instruktion	Typ	P	Q	Kommentar
ODD				test for odd
ORD	*			convert to integer
RET	*			return from block
SBI				perform integer subtraction
SBR				perform real subtraction
SGS				generate singelton set
SQI				square integer
SQR				square real
SRO	*		*	store at base level address
STO	*			store indirect
STP				stop
STR	*	*	*	store
TRC				truncate
UJC				error in case statement-abort
UJP			*	unconditional jump to Q
UNI				perform union of sets
XJP			*	indexed jump to Q+stack top

Slutligen följer ett exempel på ett enkelt Pascal-program och den P-kod som genereras. Kommentarer efter ";" är tillsatta i efterhand för att hjälpa till vid läsningen. Observera att huvudblocket i Pascalprogrammet behandlas som en procedure i P-koden. Exekveringen kommer alltså att börja på rad fem ifrån slutet.

Programmet i Pascal:

```
program EXOR;
var   X,Y,Z: boolean;
function XOR( A,B: boolean ): boolean;
begin
   XOR:= (A and not B) or (not A and B);
end; (* XOR *)
begin
   Z:=XOR(X,Y);
end.
```

Genererad P-kod:

```
 .TITL EXOR            ; programmets namn
 .ENT PCODE            ; entry point för huvudprogrammet
 .TXTM 1
 .RDX 10
 .NREL

 .ENT L3               ; entry point för XOR

L    3:                ; här börjar XOR
 ENT   1  L   4        ; "enter block" för statiskt minnesbehov
 ENT   2  L   5        ; "enter block" för dynamiskt minnesbehov
 LODB  0     12        ; ladda boolean från lokal adress 12
 LODB  0     14
 NOT                   ; invertera värdet överst i stacken
 AND                   ; gör AND på de två översta elementen i stacken
 LODB  0     12
 NOT
 LODB  0     14
 AND
 IOR                   ; gör OR på stackens två översta element
 CHKB  0      1        ; kontrollerar att värdet blev >=0 och <=1
 STRB  0      0        ; lagra funktionsvärdet vid lokal adress 0
 RETB                  ; "return boolean" - återhopp från function
L    4=      16        ; statiskt minnesbehov (används av ENT ovan)
L    5=      42        ; dynamiskt minnesbehov

 .ENT L6

L    6:                ; huvudprogrammet behandlas som en procedur
 ENT   1  L   7
 ENT   2  L   8
 MST          0        ; "mark stack"
 LDOB        40        ; ladda X, aktuell parameter till XOR
 CHKB  0      1
 LDOB        38        ; ladda Y
 CHKB  0      1
 CUP   4  L   3        ; "call user procedure" vid L3 (XOR)
 CHKB  0      1        ; kontrollera värdet XOR lämnat
 SROB        36        ; lagra resultatet vid adress 36 (=Z)
 RETP                  ; "return from procedure"
L    7=      42        ; huvudprogrammets statiska minnesbehov
L    8=      28        ; huvudprogrammets dynamiska minnesbehov
PCODE:                 ; ***** HÄR BORJAR EXEKVERINGEN *****
 MST          0        ; "mark stack"
 CUP   0  L   6        ; "call user procedure"
 STP                   ; stop
 .END                  ; programmets fysiska slut
```

3 PASCAL FÖR PC-SYSTEM

3.1 Introduktion

Pascal för en enbitsdator - går det? Det är inte ovanligt att man stöter på en smått misstänksam attityd när ämnet förs på tal. Naturligtvis kan inte själva kompilatorn köras på PC-maskinen, utan måste köras på en värd-dator. Den genererade PC-koden får därefter "tankas ner" till PC-systemet. Programutveckling kan således ske på en minidator (eller kanske till och med mikrodator) där kraftfulla programmeringshjälpmedel är tillgängliga. PC-systemet belastas således inte med någon systemprogramvara utan det genererade PC-programmet kan köras "naket" i maskinen.

Hela kedjan av operationer, från källkoden i Pascal till exekvering, åskådliggörs i figur 3.1. Cirklarna i figuren representerar in- och ut-data från de olika passen, medan rektanglarna motsvarar någon form av processorer som opererar på data. Källkoden kompileras av P-kompilatorn, som genererar P-kod. P-koden blir indata till kodgenereringspasset, ibland refererat som PBSGEN i rapporten.

Eftersom objekt-maskinen, i detta fall PBS-mini, inte har motsvarande egenskaper som den hypotetiska dator P-koden är konstruerad för, kommer delar av koden att evalueras redan under kompileringen, s k partiell evaluering Ä10,11Å. Vilka element av koden som evalueras vid kompileringen, och vilka som ger upphov till PC-kod framgår klart av följande delkapitel. Partialevalueringen tjänar huvudsakligen två syften. Dels försöker kompilatorn evaluera alla uttryck för att inte lägga ut onödig kod, s k "lazy evaluation". Dels evalueras (eller interpreteras) de språkelement som inte har någon motsvarighet i PBS-mini. Eftersom PBS-mini inte har någon hopp-instruktion kommer en for-slinga i Pascal att evalueras redan vid kompileringen och ge till resultat att kod läggs ut motsvarande antal gånger som for-slingan skulle genomlöpts. Det kan påpekas att denna evaluering inte på något sätt strider mot Pascals syntax eller semantik, utan bara är en fråga om hur implementeringen utförs.

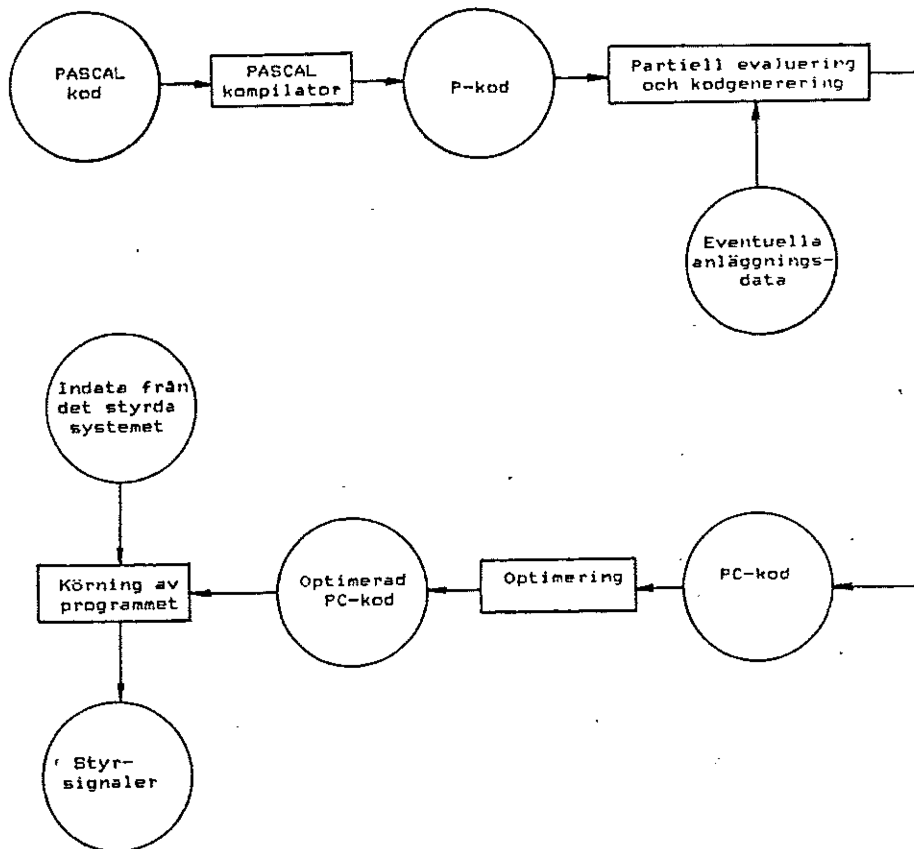


Fig 3.1

Under kodgenereringen kan data läsas in som används för att styra kodutläggningen (fig 3.1). Det är således möjligt att skriva ett generellt program för att styra en viss anläggningstyp och senare vid kodgenereringen slutgiltigt bestämma exempelvis hur många enheter av en viss komponent anläggningen ska innehålla, adresser på in-ut-gångar etc. Pascalprogrammet kan utformas så att denna konfigurering av koden m a p anläggningsdata, sker i dialog med systemet under kodgenereringen. Utdata från kodgenereringspasset är PC-koden.

För att förenkla utformandet av kodgeneratorn, och undvika alltför avancerad programflödesanalys läggs en del redundant kod ut. Denna överflödiga kod optimeras bort. Optimeringspasset, som på vissa ställen i rapporten kallas PBSOPT, känner även igen vissa kodsekvenser och byter ut dem mot specialinstruktioner som finns tillgängliga hos PBS-mini.

Koden består nu av mnemoniska koder och absolutadresser, enligt det format som presenterats i kapitel 1. Koden måste "packas" i ett binärformat och "tankas ner" i PBS-mini. Programmet exekveras därefter med insignalerna från det styrda systemet som indata, och styrsignalerna som utdata. Arbetet att packa koden och föra över den till PBS-mini är inte utfört. Ett antal provprogram har dock skrivits i Pascal. De har kompilerats och resultatet har jämförts med program skrivna direkt i PBS-kod för att verifiera riktigheten i kodgenereringen. Att binär-packa koden och överföra den till PC-systemet torde dock vara trivialt.

Hela kodgeneratorn är skriven i Pascal och ansluter till språkdefinitionen i Pascal user manual and report, Jensen & Wirth 1974. Programmen torde därigenom vara relativt portabla. Kodgeneratorn utvecklades först på en Data General Eclipse S/200. Några smärre modifieringar har gjorts för att möjliggöra körning av programmen på Digital Equipment VAX 11/780.

3.2 Önskemål på språk för PC-system

Vad har man då för anspråk på ett språk för programmering av PC-system? Ett språk på ganska hög abstraktions-nivå, med en logisk struktur som är lätt att genomskåda, torde vara att föredra. Programmeraren ska inte behöva bekymra sig om register, absolutadresser eller minnesallokering, utan kompilatorn ska administrera dessa saker. Språket ska kännas naturligt att använda, vara lättläst och automatiskt ge en god dokumentation av programmet. Program skrivna i språket får absolut inte likna krypton.

Ett PC-programs viktigaste beståndsdel är boolska tilldelningssatser och uttryck. Att dessa satser ska kunna skrivas som vanliga tilldelningssatser, från vänster till höger, med normalt vedertagna prioriteter och ett (nästan) obegränsat antal parentesnivåer, är i det närmaste självklart. Att minnesceller och in-ut-gångar ska kunna ges i sammanhanget meningsfulla namn är också klart. Man måste även i språket kunna koppla ett variabelnamn till en fysisk adress på in eller utgångar. Variabler och data bör kunna sammanfogas till strukturer för att underlätta hanteringen av dem, eller för att ge en mer verklighetsnära bild av det styrda systemet.

Villkorlig evaluering av vissa kodsegment är önskvärt. Någon form av macro-facilitet är praktiskt att ha tillgänglig. Om många liknande kodsekvenser ska skrivas är det tids- (och tålamods-) besparande att kunna definiera ett macro. Parametrar till macrot för styrning av kodutläggningen bör också finnas. Om en kodsekvens ska läggas ut många gånger kan någon slags repetitions-element underlätta.

Vid sekvensstyrningar vill man i regel dela upp problemet i mer eller mindre oberoende sekvenser som löper parallellt. Dessa sekvenser, eller processer, ska i så fall ha lokala data och regler för hur de ska interferera med sin omgivning, t ex för synkronisering eller utbyte av data med andra processer.

3.3 Implementering av Pascal för PC-system

Pascal innehåller program-element avsedda för vad som normalt går att implementera på en dator. En PC har, i förhållande till en dator, en ganska begränsad instruktionsrepertoar och i regel en outvecklad adresseringsteknik. De flesta PC kan endast representera data i logisk form, d v s motsvarande Pascals datatyp boolean. Mindre PC-system har kanske enkla "skip"-villkor för styrning av programflödet, medan större varianter tillåter hopp i programkoden eller kanske till och med subrutinhopp.

Alla dessa skiljaktigheter mellan olika hårdvaruarkitekturer gör naturligtvis att implementeringen av ett språk som Pascal kan göras i olika omfattning. Utifrån målmaskinens egenskaper får man välja ett lämpligt subset av Pascal att implementera. Följande delkapitel beskriver hur vi valt att implementera Pascal på PBS-mini och kan ses som ett förslag till hur man kan gå till väga. Många språk-element som inte har någon motsvarighet i PBS-mini har vi valt att använda för styrning av kodutläggningen, dock aldrig i konflikt med språkets definition.

Som exempel kan tas procedurer, som normalt implementeras med "jump-to-subroutine" instruktioner i en dator. Eftersom någon motsvarande instruktion inte finns tillgänglig i PBS-mini, har vi valt att expandera procedurerna på samma sätt som "macros" i en vanlig assembler. Ett anrop av en procedur i programkoden kommer alltså ge upphov till att kod läggs ut motsvarande procedurkroppen med aktuella parametrar. På så vis ändras varken Pascals syntax eller semantik. Endast sättet att implementera vissa språkelement skiljer sig från gängse sätt. Ska Pascal implementeras på en annan PC kommer implementeringens utseende i hög grad bestämmas av PC:ns hårdvaruegenskaper.

3.4 Pascal för PBS-mini

Den flitige läsaren har vid det här laget tillgodogjort sig kapitel 1, där PBS-mini beskrivs ingående, och förutsättes därmed vara väl förtrogen med dess egenskaper (och egenheter). I detta avsnitt beskrivs hur de önskemål på programmeringsspråk för PC-system, som framlades i föregående avsnitt, har tillgodosetts genom användandet av Pascal. Eftersom Pascal innehåller vissa språkelement som inte har någon relevans i PBS-mini, begränsas språket till en ren delmängd av Pascal.

3.4.1 Datatyper

Den enda typ av data som finns representerad i PBS-mini är boolean. Typen integer har tagits med i den för PC-programmering utnyttjade delmängden av Pascal, men kommer inte ge upphov till någon kod. Integers används istället för att styra kodutläggningen. Detta medför den viktiga slutsatsen att alla integers eller uttryck av integers måste vara evaluerbara vid kompileringen. Det finns flera skäl till att integers har tagits med. För att kunna implementera arrays måste integers användas som index. Att strukturera data som "array [1..limit] of boolean" kan i många sammanhang vara fördelaktigt. Integers används dessutom som styrande variabler i repetitionssatser. Integers kan även skickas som parametrar till procedurer för styrning av kodutläggningen där. För att strukturera data på ett naturligt sätt kan records utnyttjas. Kapslade strukturer av records och arrays av records kan bildas. Ett litet exempel som illustration:

```
type  motor=record
        driftssignal,
        indikeringslampa,
        tillknapp,
        franknapp: boolean
      end;

var   motorer: array[1..5] of motor;
      i: integer;

begin
   for i:=1 to 5 do motorer[i].driftssignal:=false;

osv...
```

Boolska variabler kan vara antingen in-ut-gångar hos PC:n, eller minnesceller för intern lagring. Om variabeln avser en in-ut-gång ska proceduren "origin" anropas för variabeln ifråga. Origin knyter en icke strukturerad variabel till en bestämd in-ut-gång, och behandlas utförligare i kapitel 4. Är inte origin anropad allokeras en minnescell för variabeln i PBS:ens dataminne.

Typer som real, set, char och pointer är inte medtagna. Egendefinierade skalära typer är ekvivalenta med integers och kan användas som styrvariabler och index till arrays. Records med variantdel kan användas under förutsättning att variabeln som bestämmer vilken typ variantdelen ska ha bara tilldelas värde en gång. Inskränkningen medför inga påtagliga nackdelar vid programmering.

3.4.2 Tilldelningssatser och uttryck

Boolska tilldelningssatser torde vara den mest förekommande komponenten i PC-program. Uttryck av typen boolean kommer att ge upphov till utläggning PBS-kod för beräkning av uttrycket. Uttryck av typen integer kommer däremot evalueras under kompileringen. Integer-uttryck kan på så sätt användas för att styra kodutläggningen. Kodgeneratorn försöker också evaluera boolean-uttryck för att där det är möjligt undvika att lägga ut onödig kod.

En tilldelningssats ser ut som vanligt i Pascal med parenteser för omprioritering av operationer. Eftersom PBS-mini bara har en inbyggd parentesnivå används en generellare metod som utnyttjar mellanlagringsceller vid kodutläggning av parentes-uttryck. Denna metod blir i PBS:en lika effektiv m a p tid och plats som PBS:ens egen parentesnivå. På så vis blir det möjligt att utnyttja ett godtyckligt antal parenteesnivåer vid programmering.

Uttryck omformas till PBS-kod med dess litet säregna "infix-notation" och inbakade prioriteter. Relationsoperatorer (<,>,etc) är inte implementerade på typen boolean. Några exempel på tilldelningsatser och genererad kod följer. Absolutadresserna i PBS-koden har här ingen relevans.

Pascal-kod	PBS-kod	
lampa:=true	SE	1750
lampa:=knapp1 <u>and</u> knapp2 <u>and</u> false	SEN	1750
Kaffe:=kronailagd <u>and</u> kaffeknappintryckt <u>and</u> muggnere <u>and</u> <u>not</u> muggfull	AD	1201
	AD	1202
	AD	1203
	ADN	1204
	SE	1205

```
STOP:= bryt1 and bryt2 or bryt3 and bryt4      AD   1017
       or bryt5 and bryt6                      AD   1020
                                               OR   1021
                                               AD   1022
                                               OR   1023
                                               AD   1024
                                               SE   1025
```

```
STOP:= bryt1 and (bryt2 or bryt3) and          AD   1022
       (bryt4 or bryt5) and bryt6              OR   1023
                                               SE   1027
                                               AD   1020
                                               OR   1021
                                               SE   1026
                                               AD   1026
                                               AD   1017
                                               AD   1027
                                               AD   1024
                                               SE   1025
```

```
STOP:= (bryt1 and (bryt2 or bryt3)             AD   1020
and bryt4 or bryt5) and bryt6                  OR   1021
                                               SE   1027
                                               AD   1027
                                               AD   1017
                                               AD   1022
                                               OR   1023
                                               SE   1030
                                               AD   1030
                                               AD   1024
                                               SE   1025
```

(Genom att välja vettiga variabelnamn i Pascal underlättas läsningen av programmet väsentligt. Läses endast PBS-koden i exempel tre på förra sidan är det svårt att genomskåda att det rör sig om kaffeframställning.)

Ett exempel på användandet av integers i uttryck (I förutsättes vara av typen integer och X,Y,A,B av typen boolean):

```
I:=5;                               (ingen kod läggs ut)

X:= A and B and (I=5);              AD      1022
                                    AD      1023
                                    SE      1024

Y:= A and B and (I<0);              SEN     1025
```

För att belysa användandet av integers som styrande för kodutläggningen går vi en smula i förväg. Låt oss säga att vi har 15 pumpar beskrivna som en <u>array</u> [1..15] <u>of</u> boolean. Vid ett skede i programmet ska alla pumpar stängas av, förutom pump 2 och pump 12 (decimalt) som ska sättas på. Konstruktionen i Pascal och motsvarande genererad PBS-kod:

```
for pumpnr:=1 to 15 do                      SEN     1
  if (pumpnr=2) or (pumpnr=12) then         SE      2
    Pump[pumpnr]:=true                      SEN     3
  else                                      SEN     4
    pump[pumpnr]:=false;                    SEN     5
                                            SEN     6
                                            SEN     7
                                            SEN     10
                                            SEN     11
                                            SEN     12
                                            SEN     13
                                            SE      14
                                            SEN     15
                                            SEN     16
                                            SEN     17
```

Integers kan skickas som parametrar till procedures och functions (som beskrivs senare) för att styra kodutläggningen för proceduren. Det är alltså viktigt att skilja mellan integer-uttryck som evalueras vid kompileringen och boolean-uttryck som ger upphov till PBS-kod.

I ett uttryck kan anrop av functions ske på vanligt Pascal-maner. Funktionen kan vara både av typen integer och boolean.

3.4.3 If-satser

If-satser används för att selektivt evaluera kodsekvenser. I datorer brukar if-then-else implementeras med "jump"-instruktioner. I PBS-mini finns dock inga hopp-instruktioner, utan endast en primitivare skip-vippa. Skip-vippan sätts villkorligt med instruktionen SS (Set Skip) och återställs ovillkorligt med RS (Reset Skip). När skip-vippan är satt evalueras inga instruktioner utan ses som tomma instruktioner. Utförligare beskrivning i kap 1. Skip-funktionen i PBS-mini är dock fullt tillräcklig för att realisera if-then-else i Pascal.

Om vi betraktar en if-sats i Pascal:

```
if villkor then A else B;
```

Satsen kan delas upp i två enkla if-satser utan else-gren.

```
if villkor then A;
if not villkor then B;
```

Detta motsvaras direkt av PBS-konstruktionen:

```
AD     villkor
SSN
   - A -
RS
ADN    villkor
SSN
   - B -
RS
```

Observera att uppdelningen av if-satserna till enkla sådana görs av kodgeneratorn. Programmeraren behöver alltså inte bekymmra sig om den saken utan använder Pascals if-then-else konstruktioner på vanligt sätt.

Kapslade if-satser kan på samma sätt uppdelas i enkla if-satser:

```
if villk1 then          <=>     if villk1 then A;
   A                            if villk1 and villk2 then B;
   if villk2 then               if villk1 and not villk2 then C;
      B                         if not villk1 then D;
   else
      C
else
   D;
```

Motsvarande PBS-kod blir:

```
AD      villk1
SSN
        - A -
RS
AD      villk1
AD      villk2
SSN
        - B -
RS
AD      villk1
ADN     villk2
SSN
        - C -
RS
ADN     villk2
SSN
        - D -
RS
```

Villk1 och villk2 kan i Pascalkoden ovan naturligtvis bestå av godtyckliga uttryck. Är uttrycket beräkningsbart vid kompileringen läggs kod ut endast för motsvarande then eller else gren.

När en if-sats med ett villkor av typen boolean påträffas vid kodgenereringen läggs kod ut för att beräkna villkoret. Därefter läggs kod ut för undanlagring av värdet i en speciellt reserverad cell (en för varje if-nivå). Värdet behövs om det skulle behövas i en eventuell else-gren. Undanlagringen medför också att villkoret till if-satsen inte oavsiktligt kan ändras inne i then-grenen med en felaktig exekvering av båda grenarna som följd. Om if-satsen inte skulle inehålla någon else-gren kommer koden för undanlagringen optimeras bort under optimeringspasset. På så vis kommer if-satserna inte ge upphov till någon onödig kod.

3.4.4 Repetitions-satser

PBS-minis arkitektur tillåter, som nu torde vara bekant, inga hopp i programkoden. Det har därför inte varit möjligt att lägga ut kod för Pascals repetitionssatser. Däremot används repetitionselementen som styrning för kodutläggningen. Vid kompileringen måste repetitionsvillkoren vara kända, och kod läggs ut lika många gånger som villkoren stipulerar. Detta gör att for-to-do satsen är mest användbar, främst då i samband med användandet av arrays.

```
for i:=1 to 10 do
begin

   - satser -

end;
```

Ovanstående medför således att koden för "- satser -" kommer läggas ut 10 gånger.

"Nollställning" av ett antal ingångar (utgångar eller minnesceller) kan enkelt ske med satsen:

```
for i:=1 to 25 do arr[i]:= false;
```

While- och repeat-satserna fungerar på samma sätt som ovan. Antalet gånger slingan ska genomlöpas måste vara bestämt under kompileringen. Dessa satsers användbarhet är dock begränsad.

3.4.5 Case

Case används i Pascal i huvudsak på skalära typer (integer eller egendefinierade) och kommer därför interpreteras under kodutläggningen. Vi har inte tagit hänsyn till case på variabler av typen boolean då if-then-else är ekvivalent.

3.4.6 Procedures och functions

Som nämndes i avsnitt 3.2 är det önskvärt att ha tillgång till någon form av macro-facilitet. Om många liknande kodsegment ska skrivas kan det vara arbetsbesparande att definiera ett macro. Eftersom PBS-mini har en hårdvaruarkitektur som inte tillåter subrutinhopp har vi använt "procedure" och "function" i Pascal som macro-facilitet. Proceduranropet kommer alltså att evalueras under kodutläggningen och kod läggas ut för procedurkroppen. Användandet av procedurer gör att det är lättare att modularisera och strukturera ett program, och gör det i regel lättare att läsa. Vid sekvensstyrningar kan olika parallella sekvenser läggas i separata procedurer med sina lokala variabler för att undvika oavsiktlig interferens mellan dem.

Parametrar till procedurerna fungerar på vanligt Pascal-maner. Både värde-anrop och referensanrop kan användas. Vid värdeanrop av en parameter av typen boolean, kommer en lokal minnescell allokeras i PBS-mini till vilken överkopiering sker av aktuell parameter. Den aktuella parameterns värde kommer inte ändras inne i koden svarande mot proceduren. Referensanropas variabeln (var i parameterlistan) kommer den formella parametern att referera till den aktuella, och eventuella ändringar av den formella parameterns värde i procedurkroppen kommer att resultera i ändring av den aktuella parameterns värde.

Skalära typer såsom integers och egendefinierade typer kan även skickas som parametrar till en procedur. Dessa parametrar kan användas i procedurkroppen för styrning av kodutläggningen i analogi med vad som nämnts i tidigare delkapitel. Både referens- och värde-anrop kan tillämpas.

Functions används på samma sätt som procedures med skillnaderna att de returnerar ett värde, och att anrop kan ingå i ett uttryck.

Några exempel:

Deklaration:

```
procedure ALARM(transducer,reset: boolean);
begin
   if transducer then
   begin
      redlamp:= true;
      bell:= true;
   end;
   if reset then
   begin
      redlamp:= true;
      bell:= false;
   end;
end;  (* - ALARM - *)
```

Anrop:

```
ALARM(boiler.temperature,resetbutton);
```

eller

```
for i:=1 to Numbofalarms do ALARM(tank[i].full,resetbutton);
```

I det nedre anropet kommer kod läggas ut Numbofalarms gånger med adress till rätt tanks givare.

Det är vår förhoppning att följande exempel belyser användningen av skalära typer som parametrar.

```
type  atype = (minor,fatal);

 ....

procedure ALARM(alarmtype: atype; transducer: boolean);
begin
   if transducer then
      case alarmtype of
         minor:
         begin
            redlamp:= true;
            bell:= true;
         end;
         fatal:
         begin
            BIGREDLAMP:= true;
            LOUDBELL:= true;
            MAINPOWER:= false;
         end;
      end;
end;  (* - ALARM - *)
```

I anropet anges som första parameter för vilken larmtyp kod ska läggas ut:

```
ALARM(minor,smalltank.full);

ALARM(fatal,bigtank.full);
```

Ett exempel på en function:

```
function exor(a,b: boolean): boolean;
begin
   exor:= a and not b or not a and b;
end; (* - exor - *)
```

Anrop:

```
lampa:= exor(knapp1,knapp2);
```

För att visa att det kan vara meningsfullt att definiera funktioner av typen integer ger vi följande exempel på en (rekursiv) funktion som konverterar decimala tal till oktala. Proceduren kan användas då man vill knyta en boolsk variabel till en viss in-ut-gång med "origin" (se systemprocedurer i följande avsnitt). Observera att ingen kod genereras, utan hela funktionen evalueras under kodutläggningen och returnerar ett resultat av typen integer, som sedan kan användas för styrning av kodutläggningen.

```
function Octal(decnumb: integer): integer;
begin
   if decnumb=0 then
      Octal:= 0
   else
      Octal:= decnumb mod 8 + 10*Octal(decnumb div 8);
end;  (* - Octal - *)
```

Anrop:

```
for i:=1 to 10 do origin(tempinput[i],octal(i));
```

3.4.7 Write och read

Write och read har ingen motsvarighet i PBS-mini. All in och utmatning sker direkt genom de variabler som knutits till en in-ut-gång genom "origin". Write och read-satserna kommer istället att interpreteras under kodgenereringspasset. Det betyder att in och utmatning sker mot terminalen från vilken kodgenereringen körs. På så sätt kan anläggningsdata som inte var kända under programskrivningen läsas in, dock endast i form av integers. Ledtexter och annan information kan skrivas ut med write. Man kan alltså konfigurera ett system i interaktion med kodgeneratorn under kodgenereringen.

Som en liten udda praktisk detalj kan man genom att ange filvariabeln PRR i write-satserna överföra kommentarer till PBS-koden från Pascal-koden. Detta kan ha betydelse genom att öka läsbarheten i PBS-koden om man, t ex då ett system ska tas i drift, måste gå in och ändra i maskinkoden (vilket absolut inte är att rekommendera).

3.4.8 Systemprocedurer

PBS-mini har vissa speciella instruktioner för timers, flanktriggning, "clear" av hela maskinen och exor. För dessa instruktioner, eller instruktionssekvenser, har vi, för att inte införa någon ny syntax, valt att införa ett antal systemprocedurer. Procedurerna måste "external"-deklareras i Pascal-programmet innan de används.

Hur procedurerna ska deklareras och deras funktion:

procedure origin(var boolvar: boolean; octaladdress: integer);
external;

Proceduren knyter en boolsk variabel till en oktal adress i PBS:en, i regel en in eller utgång.

function pbsaddrof(var boolvar: boolean):integer; external;

Ger den allokerade oktala absolutadressen för en boolsk variabel.

procedure clearpbs(condition: boolean); external;

Nollställer minnesceller och utgångar, och avbryter exekveringen då villkoret blivit sant. Återupptar exekveringen då villkoret blir falskt igen.

function exor(expression: boolean;var variable: boolean);
external;

Genererar PBS-instruktionen EX för variable och expression.

function edge(expression:boolean;var statevariable:boolean);
external;

Ger true då en förändring av tillståndet för expression sker från false till true. Det föregående tillståndet hålls i statevariable. Edge används för "flanktriggning" av signaler.

procedure timer(inexpr,holdexpr: boolean; var timebase,
outvar:boolean; waittime: integer); external;

Ett anrop av timer ger upphov till kodutläggning av en sekvens instruktioner för en PBS-timer. En timer används då man vill fördröja en händelse en bestämd tid. Inexpr är det uttryck som startar timern. Holdexpr är villkor för tillfälligt stopp av timern. Timebase anger tidbasen för timern. Outvar är timerns utgång och waittime anger antalet tidsenheter timern ska gå.

4. BESKRIVNING AV KODGENERATORN

4.1 Översikt

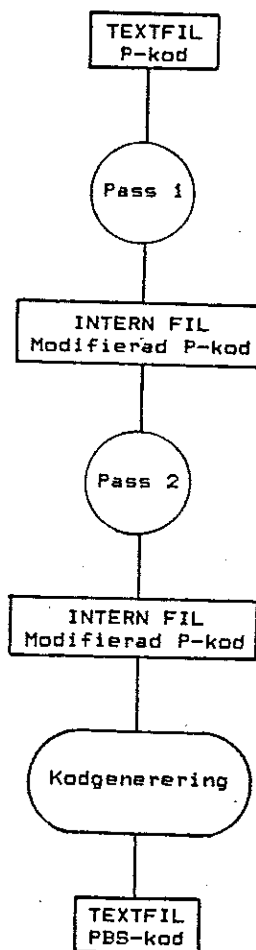
P-kodskompilatorn som används kommer från Lancaster University och är en modifikation av den ursprungliga kompilatorn från Zurich. Lancaster-kompilatorn är utvecklad till Data General Nova. P-kompilatorn körs i ursprungligt skick, d v s inga modifieringar har behövt göras för att generera PBS-kod utifrån P-koden. Kompilatorn är, precis som kodgeneratorn skriven i Pascal, vilket gör att man med en begränsad arbetsinsats kan flytta systemet mellan olika datortyper. Att flytta systemet från en DG Eclipse till en DEC VAX visade sig gå utan större mankemang.

Vi har haft som mål att skriva kodgeneratorn strukturerad och modulariserad, så att det ska vara möjligt att ändra i den relativt lätt, för att göra det möjligt att generera kod för en annan PC än PBS-mini.

Kodgeneratorn arbetar i tre pass, d v s P-koden gås igenom sammanlagt tre gånger. De två första passen går igenom P-koden och modifierar den något, samt samlar information om koden. Därefter sker den egentliga kodgenereringen.

4.2 Pass 1 och 2

Det första passet går igenom P-koden en gång och omvandlar den till ett internt format. Varje instruktion som ursprungligen består av ett antal tecken omvandlas till en egendefinierad skalär typ som skrivs ut på en intern fil.



Adresser och referenser till minnesstorlekar räknas om eftersom P-kompilatorn lägger ut kod som om boolean och integer tar två ord av minnet i anspråk. Textkonstanter i programmet lagras undan i en lista. Dessutom lagras information om vilken typ functions är av.

Pass två är till för att identifiera if-satser. If-satserna har här litet av en särställning bland Pascal-satserna. Som beskrivits i kap 3 ska en if-sats ge upphov till PBS-kod om villkoret innehåller någon boolsk variabel eller något boolskt funktionsanrop och dess värde inte är bestämt under kompileringen.

Är fallet inte så ska satsen interpreteras under det sista passet. Pass två identifierar alltså if-satser i den modifierade P-koden och tar reda på vilken typ variabler och funktionsanrop som ingår i villkors-uttrycket har. En if-then-else-sats har följande utseende i P-koden:

```
< instruktioner för beräkning av villkors-uttrycket >

FJP     L 1    ; False JumP, hoppa till L1 om värdet överst
                 i stacken är falskt

< then-grenen >

UJP     L 2    ; Unconditional Jump, hoppa till L2
L1:

< else-grenen >

L2:
```

Om denna sekvens detekteras med ett villkorsuttryck bestående av endast boolean-variabler eller anrop av boolska funktioner kommer kodsekvensen omformas till:

```
IF

< instruktioner för beräkning av villkors-uttrycket >

THEN

< then-grenen >

ELSE

< else-grenen >

EIF
```

Denna transformering görs av två skäl. Dels för att ge kodsekvensen en mera symmetrisk struktur för att senare kunna behandla kapslade if-satser rekursivt vid kodgenereringen. Dels måste man vid generering av kod för if-satser känna till om det är ett villkorsuttryck eller ett "vanligt" uttryck man är i begrepp att behandla.

4.3 Kodgenereringspasset

När pass ett och två bearbetat P-koden genereras PBS-kod. Instruktionerna läses en och en, och beroende på typ av instruktion och dess argument så genereras motsvarande PBS-kod, eller så interpreteras instruktionen. Vissa instruktioner ger endast information till kodgeneratorn och ger inte direkt upphov till någon kod. I programkoden för kodgeneratorn utförs bearbetningen av instruktion för instrukion i proceduren compile. Compile läser in en instruktion anropar en procedur för just den inlästa instruktionen via en stor case-sats. Vid behandlingen av if-satser kommer compile och if-sats-proceduren att anropa varandra rekursivt. Detta gäller även proceduren för procedures.

För att undvika att lägga ut redundant kod används ett stegs look-ahead i P-koden. Detta används framförallt vid kodutläggning av if-satser där vissa typer av kapslade strukturer kan upptäckas. Koden kan då komprimeras avsevärt. Den genererade PBS-koden läggs ut på en textfil som senare kan optimeras med ett helt separat pass för att erhålla effektivare kod.

4.4 Kodgeneratorns stack och behandling av uttryck

Den hypotetiska dator som P-koden är konstruerad för är starkt stack-orienterad. Logiska och aritmetiska beräkningar utförs överst på stacken, och är i P-koden representerade i omvänd polsk notation (RPN). Varje procedur-anrop ger upphov till att ett nytt block allokeras överst på stacken.

PBS-mini kan däremot inte ens med väldigt god vilja kallas stackorienterad. Ett av de grundläggande problemen vid generering av PBS-kod från P-kod var alltså att omvandla P-kodens stack-orienterade kod med uttryck i RPN till PBS-mini:s register-minnes struktur med uttryck representerade i infix form, inbyggda prioriteter, och parenteser i en nivå.

Detta gör att det i kodgeneratorn måste finnas en stack som _simulerar_ P-kods-datorns stack. Man kan säga att kodgeneratorn fungerar som en interpretator av P-koden som samtidigt genererar PBS-kod för instruktioner som verkar på boolska variabler och värden.

Stacken måste ha olika typer av element beroende på om en viss instruktion ska interpreteras eller översättas till PBS-kod. Dessutom måste elementtyper för representation av procedur-block, s k Mark-STack-block (MST-block), finnas.

Stacken representeras i kodgeneratorn som en "array of records" där varje record har en variantpart beroende på vilken typ den representerar. De viktigaste typerna av element är:

VALue - När en integer laddas upp i stacken under kodgenereringen skall dess värde vara känt. Värdet kan laddas upp i stacken för vidare beräkningar.

ADDRess - Om en boolsk variabel laddas upp i stacken är dess värde ibland inte känt (det ska ju beräknas under run- time i PBS:en). Istället laddas adressen upp för att kunna lägga ut kod senare med rätta adresser.

EXPRession - När en operation utföres på två element av typen ADDR länkas operander och operator ihop till en linjärt länkad lista från EXPR-elementet. Operander och operator länkas ihop i infix-ordning.

MST - Mark-stack element. Markerar att ett nytt block börjar och innehåller information om statisk och dynamisk fader till blocket samt returadress och ev returvärde från funktioner.

Genom att länka samman uttryck i form av en lista ut ifrån ett EXPR-element ger flera fördelar. Dels är det ett smidigt sätt att ,allt eftersom P-koden läses in, omvandla RPN-notation till infix-notation med prioriteter. Listan kan sedan lätt manipuleras under fortsatta beräkningar. En operation på två EXPR-listor ger upphov till en ny EXPR-lista. Sammanlänkningen kan ske på ett sätt som ger kompakt PBS-kod. Kod för undanlagring av mellenresultat vid beräkning av parentes-uttryck kan lätt länkas in i listan. Dels läggs ingen onödig kod ut om uttrycket, efter ett antal operationer, skulle visa sig vara beräkningsbart redan under kodutläggningen. EXPR-listan kan då skippas och eventuellt kan kod läggas ut motsvarande det beräknade uttrycket. Detta är viktigt eftersom P-koden evalueras partiellt, och redundant kod inte bör läggas ut av effektivitetsskäl.

För att belysa stackens funktion vid översättning av ett uttryck följer nedan ett (enkelt) exempel.

Följande sekvens P-kod ska översättas:

```
(1)  LDOB  36 ; Ladda global boolsk variabel med adress 36
(2)  LDOB  38 ; Dito med adress 38
(3)  AND      ; Gör AND på stackens två översta element
(4)  LDOB  40
(5)  LDOB  42
(6)  AND
(7)  IOR      ; Gör OR på stackens två översta element
(8)  SROB  44 ; Lagra resultatet på adress 44 (globalt)
```

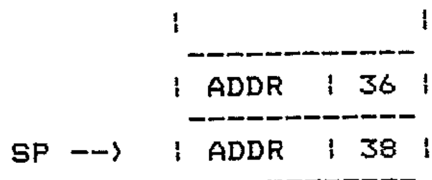
För att illustera skeendena i stacken under beräkningen betraktar vi ett antal "snapshots" efter exekveringen av varje instruktion.

(1)

```
                 !            !
                  ------------
       SP -->    ! ADDR  ! 36 !
                  ------------
```

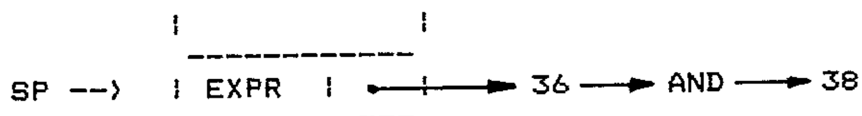
Load boolean ger upphov till ett element av typen ADDR i stacken. Adressen (36) finns med i variantdelen av elementet.

(2)



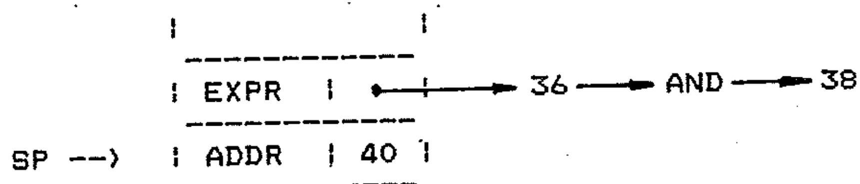
Ännu ett element av typen ADDR har laddats upp i stacken.

(3)



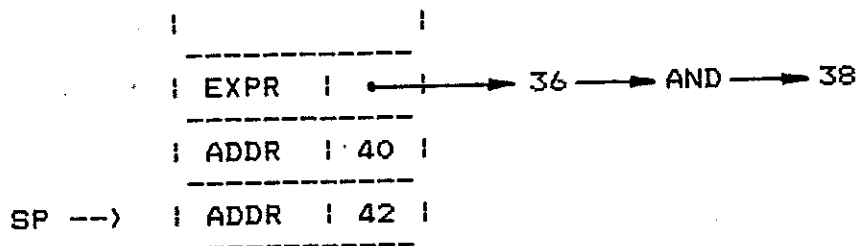
När operationen AND påträffas i P-koden länkas operationen ihop med operanderna till en lista med koden i infix notation. Stack-elementet får typen EXPR.

(4)



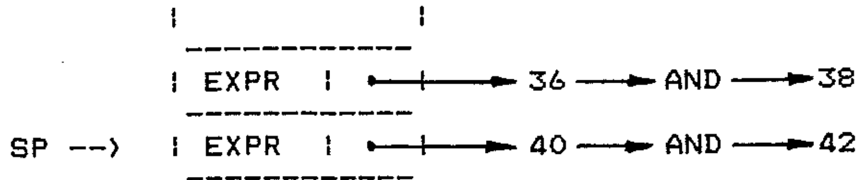
Load boolean - ett element av typen ADDR laddas upp.

(5)



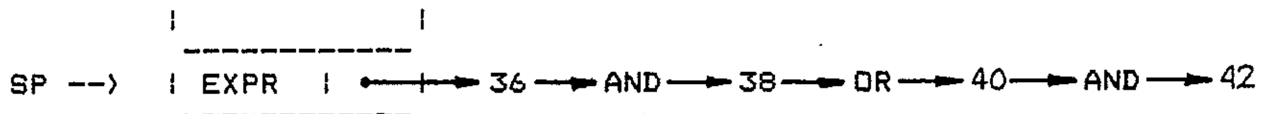
Och ännu ett.

(6)



Operationen AND utförs.

(7)



De två listorna länkas ihop till ett element av typen EXPR. I det här fallet är det inte aktuellt med någon manipulering av listan eftersom uttrycket direkt motsvarar prioritetsordningen i PBS-mini.

(8)

Store boolean vid global adress 44. Denna instruktion avslutar uttrycket och följande sekvens PBS-instruktioner genereras med EXPR-listan som grund.

```
AD      [36]
AD      [38]
OR      [40]
AD      [42]
SE      [44]
```

Hakparenteserna markerar att det i själva verket genereras instruktioner med adresser som allokerats till PBS-mini:s minne eller in-utgångar. Detta beskrivs närmare i ett senare delkapitel.

När ett uttryck behandlas i kodgeneratorn, ligger det lagrat som en lista som är bunden till ett stackelement. Denna metod valdes för att det skulle vara enkelt att hantera uttrycket med avseende på PBS-mini:s instruktions-prioriteter, AD går före OR vid evalueringen av uttryck i PBS-mini. Dessa prioriteter har ingen motsvarighet i P-koden. Det sätt på vilket vi har löst problemet med parentesnivån ger liknande problem, det gör att instruktioner oftast måste läggas till i slutet på en uttryckslista. Likaså sammanslagningen av två uttryck innebär problem.

För att lösa detta på ett praktiskt sätt har vi infört två variabler som hör ihop med typen 'expr', andlegal och orlegal. Värdena de kan anta är: noend, lastend och both (egendefinierad skalär typ i Pascal). Deras innebörd framgår av en figur längre ner.

Märk att dessa variabler endast hör ihop med boolska uttryck och därmed endast operationerna and och or. Det är när and och or ska tillämpas på en av följande kombinationer: 'expr' <op> 'expr', 'expr' <op> 'addr', 'addr' <op> 'expr', som andlegal och orlegal kommer till användning. Operanderna ligger överst på kodgeneratorns stack.

Andlegals och orlegals värden bestäms av hur det går att tillämpa ovanstående operationer på det uttryck de är relaterade till. Här nedan visas de fall som är möjliga. Observera att orlegal aldrig antar värdet noend.

...	...		
SE <arg>	SE <arg>	AD <arg>	AD <arg>
AD <arg>	AD <arg>	AD <arg>	OR <arg>
OR <arg>	AD <arg>		
Andlegal:			
noend	lastend	both	noend
Orlegal:			
lastend	lastend	both	both

Varför problemet uppstår vill vi illustrera med ett exempel. Pascal-uttrycket vi ska behandla är:

(A <u>or</u> B) <u>and</u> C

I P-kod ser det ut som följer:

```
LDOB   A
LDOB   B
IOR
LDOB   C
AND
```

Om det genereras PBS-kod för detta precis som det står så fås:

```
AD   A
OR   B
AD   C
```

Detta är dock inte ekvivalent med det ursprungliga uttrycket utan det Pascal-uttryck som PBS-koden ovan visar är:

A _or_ B _and_ C

Det riktiga PBS-uttrycket är:

```
AD   A
OR   B
SE   H   (hjälpcell)
AD   H
AD   C
```

På samma sätt kan de andra fallen illustreras. Märk att för detta senaste PBS-uttrycket har både andlegal och orlegal värdet lastend.

4.5 If-satser

De if-satser som ska översättas till PBS-kod har identifierats och omvandlats till IF - THEN - ELSE - EIF instruktioner av pass 2. Det går alltså att i förväg detektera att ett uttryck till en if-sats kommer. PBS-instruktionen RS (Reset Skip) genereras alltid vid ingången till en if-sats, eftersom villkorsuttrycket annars inte skulle evalueras om skip- vippan redan var satt.

Efter IF-instruktionen följer villkors-uttrycket i P-koden. Det behandlas på samma sätt som andra uttryck. Skulle uttrycket vara evaluerbart redan nu, kommer kod läggas ut direkt för antingen then- eller else-grenen av if-satsen, beroende på villkorets värde.

Efter villkorsuttrycket kommer instruktionen THEN som markerar början på then-grenen. Kodgeneratorn markerar att vi är inne i then-grenen på aktuell if-nivå i en array där then eller else status markeras för varje kapslad if-nivå. Kod läggs ut för beräkning av villkorsuttrycket och för undanlagring av villkors-värdet i en i PBS:en reserverad minnescell (en för varje if-nivå). För att koden i then-grenen ska exekveras vid run-time ska villkoret för aktuell nivå och alla övre liggande nivåer vara uppfyllt (jämför kap 3.4.3). Alltså genereras en sekvens:

```
AD(N)   - cell för villkorsvärde på nivå 1 -
AD(N)   - cell för villkorsvärde på nivå 2 -
AD(N)   - ...
...
SSN
```

SSN sätter skip-vippan villkorligt. AD genereras om motsvarande nivå befinner sig i then-tillstånd och ADN om den befinner sig i else-tillstånd. Därefter översätts följande kod på vanligt sätt tills en eventuell ELSE instruktion påträffas. Status i array:en för then-else tillstånd uppdateras och kodsekvensen RS - AD(N) - AD(N) -...- SSN läggs ut.

När satsen EIF påträffas är aktuell if-sats slut, och RS genereras.

I programkoden för kodgeneratorn anropas en procedur, ifproc, varje gång en if-sats upptäcks. Genom att if-satserna är symmetriskt kapslade kan ifproc anropas rekursivt när en if-sats på lägre nivå påträffas.

4.6 Procedures och functions

När P-koden genereras så kommer varje procedure att placeras separat och får också ett nytt namn i form av en label i P-koden. En egenhet med P-koden är att huvudprogrammet genereras som en procedure, vilket medför att P-kodsprogrammet börjar med ett procedure-anrop. Detta har visat sig vara en fördel vid behandlingen av programmet i vår kodgenerator, all behandling blir generell på det sättet, inga undantag behöver göras för just huvudprogrammet. Ett exempel på P-koden för ett Pascal-program med två procedurer finns i fig 4.1.

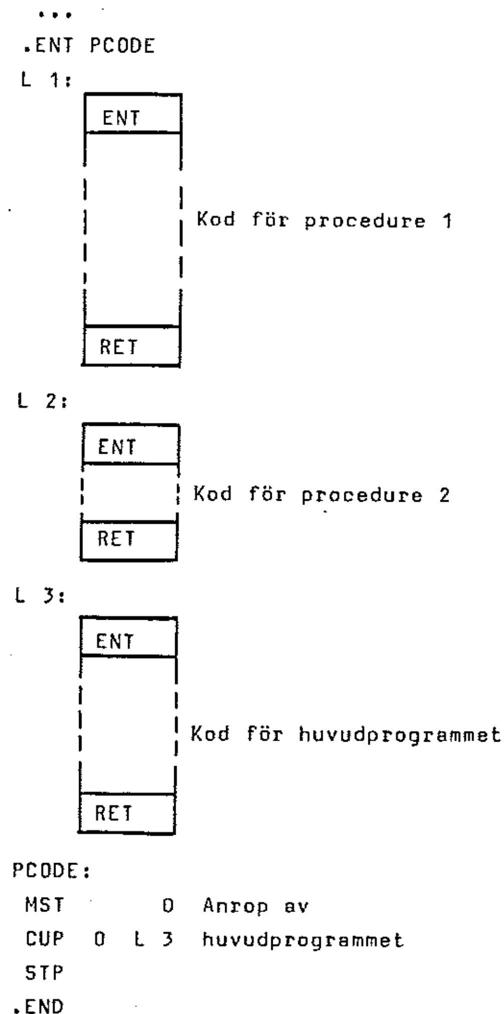


Fig 4.1
Exempel på Pascal-program med två procedurer

Vid betraktande av P-koden upptäckes att den är mycket regelbunden till sin natur. Därför lämpar det sig att vid ett procedure-anrop låta kodgenererings- proceduren anropa sig själv, dvs utnyttja rekursivitet (se fig 4.2). Då kommer ett procedure-anrop i P-koden att motsvaras av att gå ner en nivå i kodgenereringsprogrammet och därmed ett återhopp motsvaras av att gå upp en nivå i kodgeneratorn. Det här sättet att utnyttja rekursivitet gör det enkelt att hantera kodgeneratorns stack och andra väsentliga data för procedurer vid anrop av och återhopp från dem.

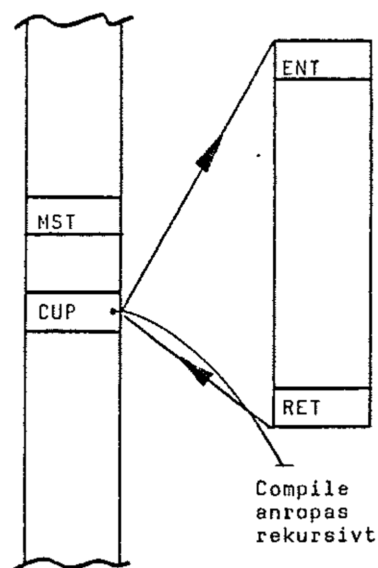


Fig 4.2
Rekursivitet i kodgeneratorn

P-koden som behandlas av kodgeneratorn finns lagrad på en intern fil. Varje instruktion tar upp en post i filen. Då vi endast läser i denna fil så har varje procedure ett bestämt läge dit vi positionerar instruktionsräknaren när den proceduren ska behandlas. All P-kod för programmet ligger alltså lagrad på en och samma fil. Man kan tänka sig andra former att organisera procedurerna t ex en fil för varje procedure e dyl.

Fyra viktiga saker att ta hand om vid ett procedure-anrop är följande:

1. Stackens utseende
2. Parametrarna
3. Procedurens minnesbehov
4. Återhoppet från densamma

För att underlätta behandlingen av dessa fyra punkter visar vi i fig 4.3 hur P-koden är organiserad för dels ett anrop och dels själva proceduren.

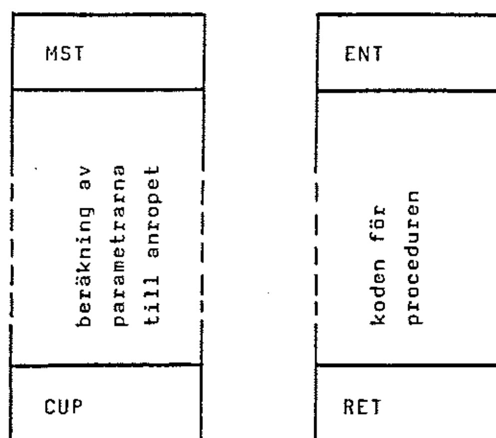


Fig 4.3
Organisationen av koden för dels ett anrop av, och dels själva proceduren.

Stackens utseende :

Till ett anrop och behandlingen av en procedure hör något som kan kallas för en miljö den behandlas i. Denna miljö består av ett block i stacken som innehåller lokala variabler och parametrarna från anropet samt pekare till det som för proceduren kan benämnas globala nivåer. De omnämnda pekarna är två till antalet och kallas allmänt för statisk och dynamisk fader.

Statisk fader är ekvivalent med vart proceduren är deklarerad och ger därmed access till alla globala variabler. Dynamisk fader motsvarar var proceduren anropades och håller alltså reda på var stackpekaren skall stå när återhoppet behandlats. Hur ett block kan se ut i stacken visas i fig 4.4.

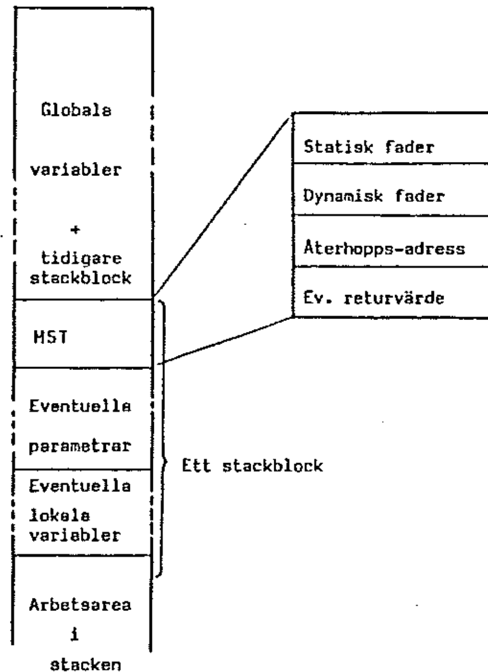


Fig 4.4
Organisationen av ett stackblock

Parametrarna :

Eftersom P-koden är tänkt att exekveras på en stackmaskin så blir genereringen av parametrarna till ett procedure-anrop ganska enkel. De läggs helt enkelt överst på stacken i tur och ordning. Koden för generering av parametrarna är placerad mellan två speciella instruktioner 'mst' och 'cup' (se fig 4.3). 'mst'-instruktionen organiserar det som beskrevs under rubriken stackens utseende. 'cup'-instruktionen motsvarar det direkta anropet av proceduren när alla parametrar är beräknade. Parameterns plats i parameterlistan bestämmer var den hamnar i stacken, relativt mark-pointern. När uttryck sänds som parameter i ett proceduranrop så evalueras det som ett vanligt uttryck och resultatet blir kvar på stacken. En del av parametrarna medför generering av PBS-kod, t ex boolska variabler "transmitted by value". De motsvaras av parameter-stackelement av typerna 'addr' och 'expr'.

Procedurens minnesbehov :

Av figur 4.1 och 4.3 framgår att i början på P-koden för en procedure finns instruktioner benämnda 'ent'. Dessa har till uppgift att reservera minne för 'mst'-elementet, parametrar och lokala variabler. Det är den första av dem som reserverar detta utrymme, den andra har ingen funktion i kodgeneratorn.

Återhoppet :

Hittills har vi ej skilt på procedurer och functions utan kallat allt för procedurer. Det finns ingen egentlig skillnad på dem utom i koden för återhoppet. Återhoppsinstruktionen avgör om det ska lämnas något värde kvar på stacken eller ej. I vår kodgenerator klarar vi av functions som lämnar antingen boolean eller integers som värde. Återhoppet har också till uppgift att rensa på stacken, och positionera instruktionsräknaren tillbaks till instruktionen efter anropet.

Vi har hitintills endast behandlat användar-deklarerade procedurer, men vi har även implementerat ett antal standard- och system-procedurer. De implementerade standardprocedurerna är följande:

```
write()        succ()
writeln()      pred()
read()         ord()
readln()
```

Systemprocedurerna är de som deklareras 'external' i Pascal-programmet. De är följande (de beskrevs även i kap. 3):

```
procedure timer();
procedure origin();
procedure clearpbs();
function exor():boolean;
function edge():boolean;
function pbsaddrof():integer;
```

Till alla dessa finns inga speciella procedur-kroppar utan det är egentligen endast ett sätt att överföra meddelande till kodgeneratorn om hur parametrarna ska behandlas och vilken slags kod som ska genereras. T ex så genererar systemproceduren timer() ett antal PBS-instruktioner som inte går att generera på annat sätt.

4.7 Adressallokering för PBS-mini

Alla boolska variabler i Pascal-programmet motsvaras av minnesceller eller in- och ut-gångar i PBS-mini. På något sätt måste då information överföras till kodgeneratorn om vilken PBS-mini-adress en boolsk variabel motsvaras av. Det kan ske dels genom systemproceduren Origin() och dels genom att låta kodgeneratorn ge variabeln en PBS-mini-adress. Det senare tillämpas lämpligen på alla de variabler som inte ska knytas till någon speciell adress.

Istället för att utnyttja PBS-mini:ns inbyggda parentes-nivå så använder vi PBS-mini-celler som mellanresultats-celler, den metoden är mycket mer generell då det gäller behandlingen av uttryck. Dessa mellanresultats-celler används endast för att överföra resultat på kortare avstånd i PBS-koden, de kan alltså användas igen för samma sak. Globala boolska variablers PBS-adresser kan användas endast till de variabler de blivit associerade till och de får dessutom ej ha varit använda innan de binds till variablerna. Orsaken är att PBS-programmet är cykliskt och variablers värden får ej förstöras.

Det finns fler ställen där det används PBS-minnesceller som kan lämnas tillbaka t ex procedurers parametrar och lokala variabler. Med hjälp av en speciell typ i Pascal håller vi reda på i vilket tillstånd en PBS-minnescell befinner sig. Alla PBS-adresser förvaras i en tabell, och vi har gjort en del procedurer för att kunna hantera den tabellen. De procedurerna utför vad som behövs vid hämtningen av en adress och en eventuell returnering av den. Systemproceduren Origin nyttjar dessa procedurer.

I kodgeneratorn finns en variabel med namnet semaphor, vilket kan misstolkas, men det rör sig inte om realtidsprogrammering i kodgeneratorn utan denna variabel används endast som skydd i samband med adressallokeringen.

Vid anrop av funktioner så returneras det alltid ett värde och för boolska funktioner så blir det även PBS-kod, här används en PBS-adress för att överföra funktionsvärdet.

Ifall en boolsk funktion anropas i början av ett uttryck och det senare i uttrycket behövs en hjälpcell för att överföra ett värde (t ex en parentesnivå e d) så kan följande inträffa: Funktionsvärdets PBS-adress återlämnas (p g a att vi återvänder från funktionen och PBS-koden är genererad) och samma adress används för hjälpcellen, med felfunktion som resultat. Detta får aldrig inträffa.

En lösning är att aldrig återlämna funktionsvärdens PBS-adresser, men ett program som består av många boolska funktioner kan få problem då, ty adressutrymmet kanske inte räcker till trots att det borde ha gjort det. Därför har vi löst det med variabeln semaphor, listan leavepool och procedurerna leavelist och pushleave.

5. OPTIMERINGSPASSET

5.1 Introduktion

En kompilator är ett deterministiskt program vars handlingar bestäms av indata och tidigare tillstånd. Om kompilatorn är korrekt, så genererar den vid körning med samma indata alltid samma kod. Ponera att vi har två program som utför exakt samma uppgift, men de är skrivna på olika sätt (exvis: strukturerat och icke-strukturerat). Vid kompileringen uppfattar inte kompilatorn att programmen är likvärdiga, den genererar kod allteftersom den läser in programmet som ska kompileras, och uppfattar därmed inte helheten. Detta är en nackdel, ty den genererade koden kan, i antal instruktioner mätt, bli väsentligen längre för det ena programmet, än för det andra. Det har ofta varit ett standard-argument från "smart"-programmerarna mot strukterad programmering, de vill hävda att strukturerad programmering ger väsentligt längre kod.

En lösning på problemet är - optimering. Genom att optimera den genererade koden så har man kvar den strukturerade programmeringens fördelar och finesser, och får samtidigt en längd på koden som är jämförbar med "smart"-programmerarens. Ännu en fördel är att man inte behöver bekymra sig, eller lära sig, något om den kod som kompilatorn genererar för de olika språkkonstruktioner som ingår i det programspråk som används. Därför är det endast högnivåspråket som behöver läras in, medan "smart"-programmeraren behöver kunna både språket och den genererade kodens utseende och struktur, för att kunna använda sin "smartness".

Idag är datorkraften en billig kraftkälla, åtminstone om den ställs i relation till programmerarkostnaden. Detta gör att vi inte har råd att sköta om saker som datorn gör lika bra eller tom bättre.

Vilka metoder som kan användas, och vilka vi har valt behandlas i följande delkapitel. Där visar vi också en del exempel och förklarar efter vilka principer vi optimerar.

5.2 Olika typer av optimering

Det finns många olika typer av optimering. Vi skall bara ta upp ett par som inledning och jämförelse till de metoder vi valt. I nästa delkapitel beskrivs hur vi gör vår optimering, och vilka antaganden mm den bygger på.

Det går att dela in optimeringsmetoderna i olika grupper. Här följer en indelning av de metoder som varit lämpliga för oss:

1: Optimerande kompilatorer - dvs optimeringen är inbyggd i själva kompilatorn.

2: Kompilatorn genererar kod som i sig inte är exekverbar, men som är på en lägre nivå än det högnivåspråk programmet var skrivet i. Denna kod är mycket lämplig att optimera och generera exekverbar kod från.

3: Kompilatorn genererar exekverbar kod som sedan blir inmatad i ett speciellt optimeringsprogram. Här kan två dominerande metoder urskiljas:

a: Byte av instruktionssekvenser mot andra, effektivare och passande instruktioner. Detta benämns ibland "peephole optimisation".

b: Dataflödesanalys: Genom att undersöka förändringar i data går det att avgöra om en viss instruktionssekvens någonsin kommer att användas. Ifall det visar sig att den aldrig användes så kan den tas bort.

Det finns både för- och nack-delar med alla metoderna, men för vår del vilar dessa på lite olika ställen i de ovan beskrivna metoderna.

T.ex. så kunde metod 1 helt ha spolierat vårt projekt, dvs i syftet att utgå från en befintlig P-kodskompilator. Vårt projekt bygger på en kompilering och därefter följande partiell evaluering av den kod kompilatorn genererade. Om nu kompilatorn hade levererat färdigoptimerad kod, hade vi kanske inte haft tillräcklig information för att utföra vår partialevaluering. Det går visserligen att konstruera fram viss information ur den givna koden (jmfr scannerpasset kap 4), men detta är en nackdel och ofta ganska tidskrävande. Bästa lösningen, när den kompilator som används är gjord enligt metod 1, är att optimeringen går att stänga av på något sätt t.ex. med en option.

Metod 2 är betydligt bättre och påminner om vår egen partialevaluering. På grund av den myckna information koden ofta innehåller så går det knappast att direkt interpretera den, utan valet blir att optimera på det stadiet, för att sedan generera exekverbar kod från den optimerade koden.

Om man överhuvudtaget kan anföra någon kritik mot metod 2 så skulle det vara att den kräver flera programpass för att göra koden exekverbar, men vi tycker inte att detta är någon egentlig nackdel utan i många sammanhang en fördel, snarare då flera pass är lättare att hantera än ett stort program.

Den typ av Pascal-kompilator som vi använder till den kompilerande delen av vårt projekt, är en s.k. P-kompilator [kap 2] och den genererar ofta en kod som har metod 2:s princip men i ett enklare utförande.

Metod 3 har egentligen stora likheter med metod 2 och kanske kan de sammanföras till en grupp av optimeringsmetoder. För oss fanns det skäl att skilja på dem i den här allmänna översikten, dels på grund av att vi tar in P-kod och genererar PBS-kod, och dels att optimeringen inte anbringas på P-koden utan på PBS-koden. De optimeringsmetoder vi beskriver här är oftast tillämpade på assemblerkod, medan vi tillämpar dem på PBS-kod som har ett enklare och något annorlunda utförande [se kap 1].

Fördelen, jämfört med t.ex. metod 2 är att koden kan köras i icke optimerat skick, dvs det är möjligt att välja om man vill offra tid på optimeringen, eller inte. En annan fördel är, jämfört med metod 1, att om det visar sig att programmet går långsamt när det körs, optimera det och det går något snabbare. Metod 1 som tex. har optionsstyrd optimering kräver då en omkompilering för att optimeringen ska göras, dvs en betydligt långsammare process. Nackdelar med metod 3 är att det tar (precis som med metod 2) relativt lång tid att få ett färdigoptimerat, program och komplexiteten hos optimeringsprogrammet har en benägenhet att bli ganska stor pga att koden ska analyseras.

Metod 3a är enklare att implementera än 3b, men den ger å andra sidan inte alltid lika bra resultat. I övrigt kommenterar vi inte skillnaden mellan metoderna 3a och 3b (Se även referenslistan).

Till slut ska sägas att de flesta optimeringsmetoder som kommer fram oftast är s.k. "hemsnickeri" och då vanligen en blandning av dessa tre och andra mer eller mindre kända metoder. Så även vår metod.

5.3 Optimeringsprinciper i PBSOPT

En berättigad fråga som kan ställas är om vi behöver använda optimering, då den kod vi genererar befinner sig på s.a.s. lägsta nivå och verkar inrymma minimal komplexitet. Det finns visserligen en del komplicerade begrepp t.ex. timers mm., men en del av dessa har vi specialimplementerat.

Det visade sig vid utvecklingen av PBSGEN att vissa saker var mer komplicerade än vi trodde i början och att lösa de problemen i PBSGEN var besvärligt. I det läget valde vi att utveckla ett optimeringspass som skulle klara av de redundanta kodsekvenser vi var tvungna att lägga ut på vissa ställen. Till en början bestod PBSOPT bara av ett fåtal olika optimeringsmönster, men med tiden visade det sig att det var bra att lägga in allt fler optimeringsmönster och metoder.

De svårigheter som omnämndes i början berodde på endast en satskonstruktion i Pascal, nämligen if-then-else. I övrigt är det endast standardfunktionerna Exor och Edge som påverkas av optimeringen. Om PBSGEN skulle innehållit programkod för att klara det vi gör med PBSOPT, så skulle mängden extra kod i PBSGEN antagligen ha blivit större än hela nuvarande PBSOPT. Möjligheten att enkelt hantera koden är också ett skäl till att ha optimeringspasset separat (jmfr kap 5.2).

Det är i huvudsak två moment vid hanteringen av if-satser i PBSGEN som orsakar de problem som nämndes ovan. Dessa beskrivs närmare i följande två punkter:

1: Vid nästning av if-satser så orsakar villkoret en hel del problem. Nästning av if-satser fungerar inte i PBS-koden utan där används specialknep för att komma runt det problemet. Hur det går till beskrivs i kapitel 4. Detta problem uppstår inte bara vid nästning utan också om villkoret till if-satsen innehåller ett funktionsanrop som eventuellt kommer att generera PBS-kod.

Vi har löst det genom att alltid generera en 'RS'-instruktion när vi börjar behandla en if-sats. Detta medför att om skip-vippan i PBS-minin (se kap 1) är satt så nollställs den, om inte så är det en tom instruktion, dvs den gör ingen skada utan ser bara ut att vara konstigt placerad, för den som är van att läsa PBS-kod. Den extra instruktionen fyller bara ut minnet och om PBS-koden optimeras så försvinner instruktionen. Ett par exempel på hur Pascal-kod som ger dessa extra 'RS'-instruktioner ser ut, finns i fig 5.1. Motsvarande PBS-kod har också tagits med för fullständighetens skull.

Pascal-kod

```
 ...
if REGISTRERING then
begin
   if not MITT2 then KORT:=true;
   if START2 then LAANG:=true;
end;
 ...
```

PBS-kod
Icke optimerad

```
...
AD  1057
SE  1004
RS  -onödig-
ADN  412
SE  1005
AD  1004
AD  1005
SSN
SE  1053
RS
RS  -onödig-
AD   411
SE  1005
AD  1004
AD  1005
SSN
SE  1055
RS
RS  -onödig-
...
```

figur 5.1
Detta utdrag ur en programlistning visar genereringen av extra 'RS'-instruktioner i samband med if-satser

Problemet uppstod när vi försökte slippa generera den instruktionen redan i PBSGEN, då skulle vi behöva göra en noggrann analys av varje if-villkor och se om det innehöll något som kunde orsaka problem och sedan special- behandla det. Denna analys är jämförbar med någon form av lexikalisk analys i en kompilator och är ganska kod-krävande, även i Pascal. Valet kunde då ha varit att förbjuda nästning av if-satser och funktionsanrop i if-villkoret, men detta vore att förstöra ett av motiven för att införa högnivåspråk för programmering av PC-system.

Genom att göra en mer krävande kodanalys i PBSGEN, så skulle vi i högre utsträckning kunna eliminera redundanta 'RS'-instruktionerna, på ett enkelt sätt redan i PBSGEN. Det lönar sig dock inte eftersom vi ändå måste ha PBSOPT kvar och vi ser det meningslöst att öka komplexiteten i PBSGEN bara för den sakens skull, när samma rutin ändå utförs i PBSOPT.

2: Nästa stora problem är att PBS-mini saknar hopp-instruktioner, ty if-then-else-konstruktionen löses vanligast och enklast med hopp. Problemet förklaras enklast med hjälp av fig 5.2.

Med kännedom om hur PBS-mini exekverar sitt program (se kap 1) så syns det att om villkoret 'A' är sant så kommer alt A/ inte att fungera som Pascal-strukturen visar. Både then- och else-grenen kommer att exekveras med de fel det orsakar, alt B/ kommer däremot att fungera precis som det var tänkt. Dock kommer detta sätt att generera extraceller för att klara exekveringen av if-satser korrekt, att ge rätt mycket extra PBS-kod, ty if-satser är den vanligaste satskonstruktionen, och då i synnerhet bara if-then utan else-gren, i Pascal-PBS-programmen. Som lätt inses så behövs inga extra celler om man har endast if-then och koden mellan then och nästa ';' inte innehåller några if-satser.

Att ta bort dessa extra celler redan i PBSGEN skulle antagligen kräva att vi tog in hela if-satsen, med eventuella nästningar och allt (dvs obegränsad look-ahead), för analys av koden och därefter bestämma var extra celler ska genereras. Hur kostsamt och besvärligt det är att göra denna analys mm, gör vi inga detaljerade försök att beskriva här, men besvärligt är det...

```
Pascal-kod            alt A/            alt B/

if A then             AD      A         AD      A
begin                 SSN               SE   1004
   ...                ...               AD   1004
   A:=false;          SEN     A         SSN
   ...                ...               ...
end                   RS                SEN     A
else                  ADN     A         ...
begin                 SSN               RS
   ...                ...               ADN  1004
end;                  RS                SSN
                                        ...
                                        RS
```

figur 5.2
Beskrivning av en if-then-else-konstruktion i Pascal
och två alternativa genereringar av PBS-kod,
som motsvarar Pascal-koden

Borttagningen av extra celler löses istället enkelt i PBSOPT med hjälp av en länkad lista, några pekare och procedurer. Beträffande detaljer i implementeringen av metoderna för optimeringen som löser de två problem vi beskrivit här, se kap 5.4, där beskrivs implementeringen av PBSOPT mera i detalj.

Ovan har vi alltså beskrivit de två stora problemen vid kodgenerering för if-satserna, och hur vi löst dem, det finns dock ännu en programbit i PBSOPT som påverkar koden som genererats för if-satserna.

PBS-mini:s instruktionsrepertoar (se kap 1) inrymmer inte endast 'SE'- och 'SEN'- instruktionerna för att påverka en specifik minnescell, det finns även två instruktioner, 'SSR' och 'RSR', som betraktar minnescellen som en SR-vippa, funktionen hos denna beskrevs i kap 1. Dessa instruktioner ('SSR'&'RSR') har en exakt motsvarighet i sekvenser av andra PBS-mini-instruktioner, se fig 5.3.

```
...                                  ...
<villkor>                            <villkor>
SSR <minnescell>     <=>             SSN
...                                  SE  <minnescell>
                                     RS
                                     ...
```

```
...                                  ...
<villkor>                            <villkor>
RSR <minnescell>     <=>             SSN
...                                  SEN <minnescell>
                                     RS
                                     ...
```

figur 5.3
'SSR'- och 'RSR'-instruktionerna och dess ekvivalenter i annan PBS-kod

Generell if-sats:

```
if <villkor> then                    <villkor>
begin                                SSN
   ...                <=>            ...
end;                                 RS
```

Två specialfall:

```
...                                  ...
if <villkor>                         <villkor>
then <variabel>:=true       <=>      SSN
...                                  SE  <minnescell>
                                     RS
                                     ...
```

```
...                                  ...
if <villkor>                         <villkor>
then <variabel>:=false      <=>      SSN
...                                  SEN <minnescell>
                                     RS
                                     ...
```

figur 5.4
Pascal-kod för generell if-sats och för två specialfall, samt den ekvivalenta icke-optimerade PBS-koden

Det skulle antagligen gå att generera 'SSR'- och 'RSR'-instruktionerna redan i PBSGEN, men som anförts tidigare, det är frågan om det är värt besväret då det ändå ska köras ett optimeringsprogram efter kodgenereringen. Den anledningen vägde tyngst när vi valde att placera Pascal-koden för optimeringen av de konstruktioner som visas i fig 5.3 i PBSOPT istället för i PBSGEN. Antagligen blev mängden Pascal-kod mindre pga lättare analys. De if-sats-konstruktioner som ger den PBS-kod som visas i fig 5.3, visas i fig 5.4.

En detalj i denna del av optimeringen är att den koden som ska optimeras har ett fast format, dvs den består alltid av tre PBS-instruktioner. Optimeringen kan då ske med ett s.k. fönster som glider över inmatningsfilen och detta fönster matchas mot fasta kodmönster, och när de är lika så görs optimeringen. Denna metod behandlade vi som metod 3a (peephole optimization) i kap 5.2, och beträffande detaljer i implementeringen så behandlas den i nästa delkapitel.

Tidigare i kapitlet angavs att PBS-koden för systemprocedurerna Exor och Edge påverkas av optimeringen. Det kan verka onödigt, de är dock systemprocedurer och då borde optimeringen klaras i kodgeneratorn. I detta fallet beror optimeringen på den omgivning i vilken proceduren anropas. Därav följer att de ägnas kraft i optimeringspasset.

För att generellt kunna använda Exor och Edge så måste det genereras en extra minnescell som innehåller funktionsresultatet, ifall anropet ingår i ett uttryck. Detta behövs då PBS-koden för en funktion inte kan förekomma mitt inne i annan PBS-kod, och hoppinstruktioner inte finns. Därför måste resultatet sparas i en cell för att senare kunna användas i uttrycket.

Pascalkod	PBS-kod			
	Icke optimerad		Optimerad	
A:=EDGE(B,C);	AD	1017	AD	1017
	EDGE	1021	EDGE	1021
	-SE	1023-	SE	1024
	-AD	1023-		
	SE	1024		
A:=<u>not</u> EXOR(B,C);	AD	1017	AD	1017
	EX	1021	EXN	1021
	-SE	1023-	SE	1024
	-ADN	1023-		
	SE	1024		

figur 5.5
Två exempel på onödig generering av extraceller pga att Edge och Exor är implementerade som funktioner. Exemplet med Exor visar även den andra optimeringen som påverkar Exor.

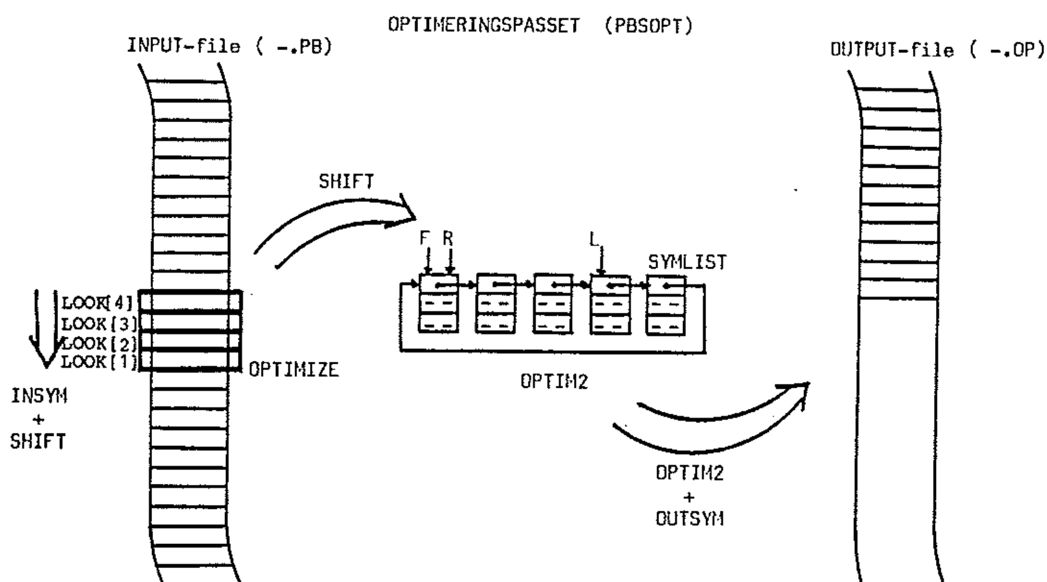
Det finns fall då denna extra cell inte behöver genereras, utan den kan elimineras. För att vara riktigt säker på att det inte blir fel när den extracellen tas bort bör det vara funktioner som man har full kontroll över som utsätts för denna optimering. Detta har vi i fallet Exor och Edge eftersom de är s.k. standardfunktioner. Exempel på när onödig kod genereras finns i fig 5.5.

De här extra PBS-instruktionerna tas bort på samma sätt som optimeringen av PBS-koden för vissa if-satser gav 'SSR'- och 'RSR'-instruktioner.

Exor-instruktionen påverkas ytterligare, då den har en variant till bland PBS-instruktionerna nämligen 'EXN'-instruktionen. Denna införs med hjälp av PBSOPT där det går att göra, dvs endast där det redan finns en 'EX'-instruktion och omgivningen till denna är sådan stt vi kan byta ut dem. Se nästa delkapitel för detaljer.

5.4 Implementering av PBSOPT

I förra kapitlet beskrev vi den allmänna utvecklingen av vårt optimeringspass. Det innehöll även en del diskussion kring kodgenerering kontra optimering. I detta kapitel ska vi mer detaljerat, och med mindre diskussion kring ämnet, behandla själva implementeringen av optimeringspasset PBSOPT.



figur 5.6
Ett enkelt flödesschema som beskriver vad PBSOPT gör

En helhetsbild av vad PBSOPT gör, fås lättast genom att studera fig 5.6, som beskriver hur PBS-koden som skall optimeras flyter genom PBSOPT. Av fig 5.6 framgår att PBSOPT består av fyra (fem) delar:

- Initiering (ej med i fig 5.6)
- Inmatning från en datafil med PBS-kod
- Optimering, del 1
- Optimering, del 2
- Utmatning till resultatfilen

Varje del utgörs av en eller flera procedurer som anropas i tur och ordning från huvudprogrammet och andra procedurer. Resten av kapitlet ägnas åt en kortfattad genomgång av varje delmoment och dess procedur(er). Förståelsen av dessa beskrivningar kan bli bättre om fig 5.7 först studeras och sedan hålls i minnet vid läsandet av beskrivningarna. En utskrift av hela PBSOPT finns i kapitel 8.

```
...
while not ENDREACHED do
begin
    INSYM;
    SHIFT;
    OPTIMIZE;
    OPTIM2;
end;
...
```

figur 5.7
Strukturen i PBSOPT:s huvudprogram

Initieringen:

Består av proceduren INITVARIABLES. Här byggs söktabellen för inkommande instruktioner, upp som en array. Alla boolska variabler och pekare initieras till lämpliga värden. Dessa används till att påbörja, styra och avsluta exekveringen på ett korrekt sätt.

Inmatningen:
Består av procedurerna INSYM, SHIFT och funktionen READCH. Infilen förväntas innehålla ett icke optimerat PBS-program, genererat av PBSGEN. Funktionen READCH är en hjälpfunktion till proceduren INSYM, den levererar nästa tecken från infilen.

INSYM läser in nästa instruktionsrad och letar upp vilken instruktion det är. Om det är kommentar-instruktionen så läses hela kommentaren in och lagras i en speciell länkad lista för kommentarer, annars läses endast eventuellt argument till instruktionen in. INSYM lämnar symbolen och dess eventuella argument i variablerna CURRSYM och CURRARG. INSYM håller också reda på när infilen tar slut så att läsning inte sker längre, detta görs med den boolska variabeln ENDTOUCHED. Som förstås av namnet på variabeln så tar exekveringen och optimeringen inte slut just när ENDTOUCHED signalerar att infilen är slut.

SHIFT:s huvuduppgift är att manipulera det s.k. fönstret som glider över infilen i fig 5.6, detta fönster används i del 1 av optimeringen. Själva manipuleringen består i att flytta fönstret ett steg längre ner på infilen. Detta görs genom att mata ut sista elementet i fönstret till den lista som används i del 2 av optimeringen, därefter flytta upp de resterande elementen en position och allra sist lägga in CURRSYM och CURRARG på den nu tomma platsen i fönstret.

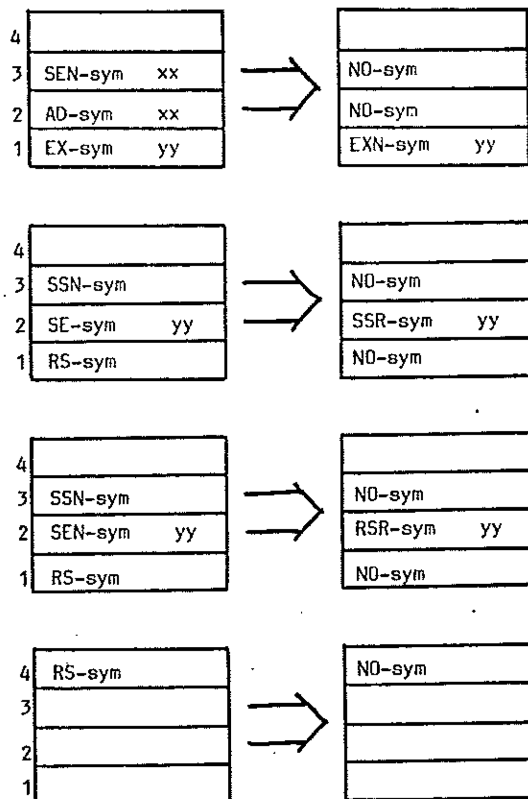
SHIFT har också till uppgift att registrera när optimeringen och exekveringen ska avslutas. När 'END'-symbolen matas ut till listan så är optimeringen slut, variabeln ENDREACHED används för att registrera detta.

I en tidigare version av PBSOPT hade vi inte med del 2 av optimeringen utan SHIFT skickade symbolerna den matade ut från fönstret direkt till resultatfilen. När vi införde del 2 så uppstod problem som senare lokaliserades till SHIFT:s överföring av symboler från fönstret till listan. Problemet bestod i att del 1 vid optimering genererar symbolen 'NOSYM' på olika ställen i fönstret, när en 'NOSYM'-symbol matas ut med SHIFT så ska den bara försvinna och inte matas in i listan. Vi registrerade inte att det inte kom in någon symbol i listan utan gjorde del 2 av optimeringen som vanligt, vid vissa speciella sekvenser av PBS-kod bar sig del 2 säreget åt. Dessa problem eliminerdes med den boolska variabeln SYMSHIFTED som registrerar om en verklig symbol skiftats ut från fönstret eller inte. Om SYMSHIFTED är sann så kommer del 2 att utföras i denna vändan av huvudprogrammet, annars står den över tills SYMSHIFTED blir sann igen.

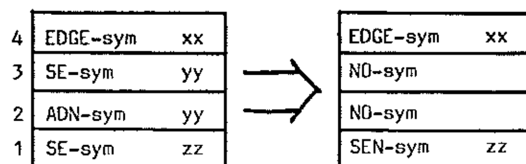
Optimering, del 1:

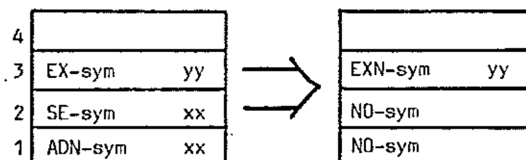
Består endast av proceduren OPTIMIZE. OPTIMIZE utför endast operationer på det s.k. fönstret som är en array av fyra records, vardera innehållande en instruktionssymbol och dess eventuella argument.

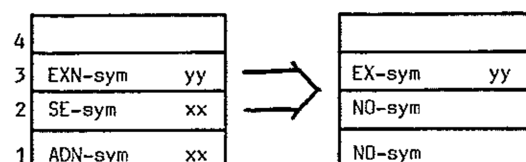
OPTIMIZE jämför fönstret med ett antal statiska mönster och optimerar ifall något mönster matchar det mönster fönstret just nu innehåller. De mönster fönstret matchas mot är uppritade i fig 5.8, och är så gott som självförklarande. Ifall något verkar oklart så får utskriften av PBSOPT i kap 8 tjäna som hjälpreda.

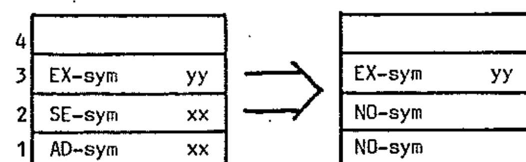


figur 5.8
På denna och nästa sida visas de optimeringsmönster OPTIMIZE använder sig av









Denna sista optimeringstyp utförs också om det finns en EDGE-sym eller EXN-sym på plats nr 3.

OPTIMIZE innehåller även en enkel dynamisk optimeringsdel, den fanns med i fig 5.8 men tarvar en liten förklaring. Det är räkningen av 'SSN'- och 'RS'-instruktioner som avses. Dessa kan inte nästas utan bara användas på en nivå, dvs vi kan plocka bort överflödiga 'RS'-instruktioner. Detta sköts med OPTIMIZE och den boolska variabeln SKIPSET som håller reda på om vi befinner oss mellan 'SSN' och 'RS' eller inte.

Optimering, del 2:

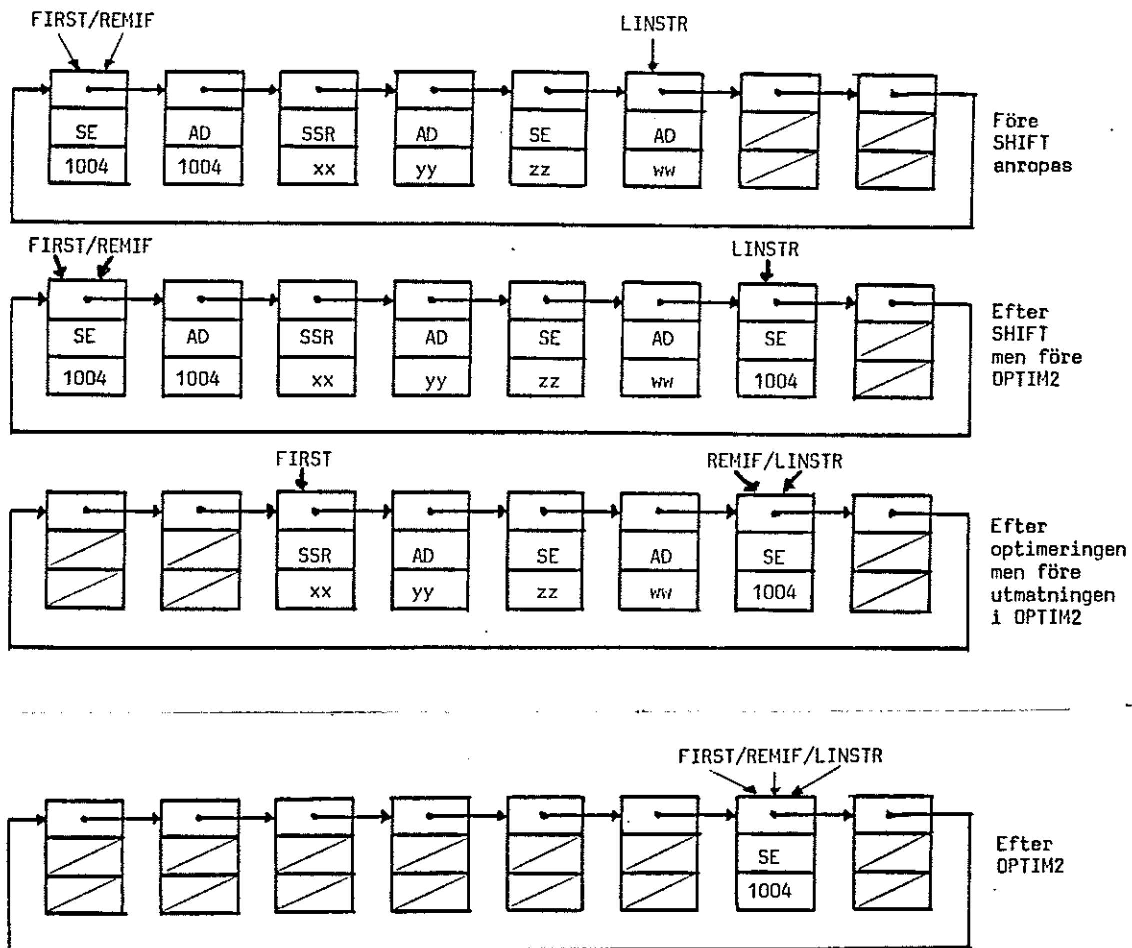
Består av första delen av proceduren OPTIM2. OPTIM2 utför en helt dynamisk optimering, den utförs på PBS-koden för if-then-else-konstruktionen och undersöker om några av de extra instruktioner som genererats för att klara if-satserna korrekt, går att ta bort. Ifall det visar sig vara möjligt så tas de bort.

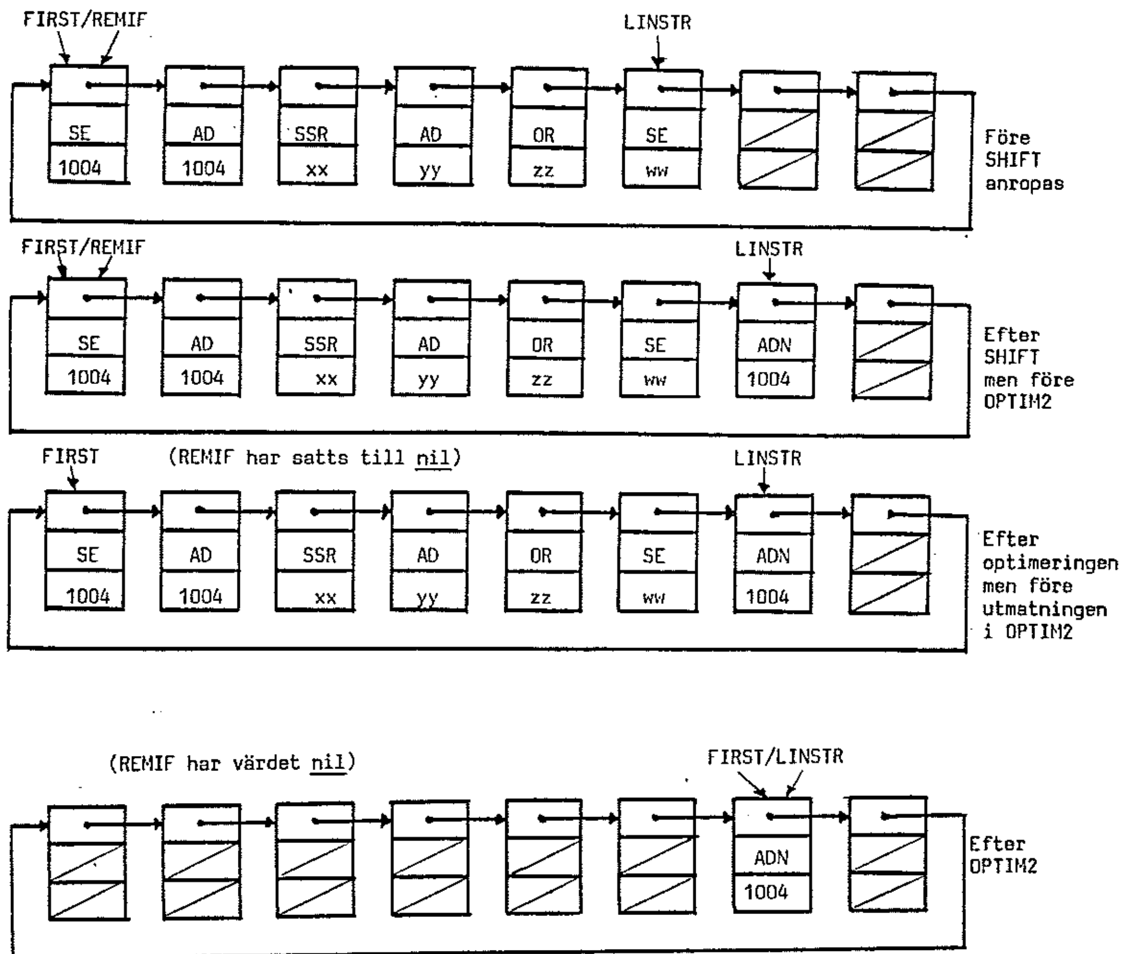
De "redskap" som används för optimeringen är tre pekare, FIRST, LINSTR, REMIF och en cirkulär enkelt länkad lista, SYMLIST. Hela anordningen fungerar ungefär som en s.k. ringbuffer. Skillnaden beskrivs på följande sätt, om det behövs mer plats i listan vid optimeringen så stoppas där in fler nygenererade noder, medan en ringbuffer vanligtsvis har en från början fastlagd och begränsad storlek.

Pekarna FIRST och LINSTR används för att begränsa listan i SYMLIST, de flyttas runt SYMLIST under exekveringens gång. De har värdet nil endast vid initieringen. REMIF som också initieras till nil byter värde, mellan nil och någon nod i SYMLIST. Den ändringen av värdet beror på SYMLIST:s innehåll. REMIF håller reda på aktuellt optimeringstillstånd, att vi använder en pekare för det ändamålet grundar sig på att optimeringstillståndet är dynamiskt.

Med optimeringstillståndet menas SYMLIST:s aktuella innehåll och REMIF:s värde. Om det först i listan förekommer instruktioner som går att optimera bort, så används REMIF för att markera detta och för att kunna ta bort dem när det är lämpligt. REMIF:s värde ändras beroende på vilka instruktioner som kommer in till SYMLIST via LINSTR. Om den instruktion som kom in till SYMLIST senast medför att vi inte kan optimera bort de instruktioner som REMIF markerar, så sätts REMIF till nil. REMIF förblir nil tills en instruktionssekvens som kan markeras för optimering dyker upp i SYMLIST.

Exempel på en sekvens av instruktioner, kombinerat med att REMIF är skild från nil, som optimering kan utföras på, visas överst i fig 5.9. I figuren visas stegvis hur optimeringen går till. Denna sekvens motsvarar som tidigare sagts if-then-konstruktionen. Det finns två typer av instruktionssekvenser som omintetgör eventuell markerad optimering, de motsvaras av if-then-else-konstruktionen och nästningen av if-satser. Exempel på den förra finns i nedre delen av fig 5.9. Dessa medför alltså att optimeringen så att säga tillfälligt stängs av.





figur 5.9
På denna och föregående sida visas i två exempel hur proceduren OPTIM2 arbetar

Då OPTIM2 är den sista optimeringsproceduren i huvudprogrammets while-slinga så måste proceduren innehålla kod som avslutar PBSOPT korrekt, dvs utför eventuell kvarvarande optimering m.m.. Denna kod är av naturliga skäl placerad sist i OPTIM2.

Utmatningen:

Består restrerande delar av OPTIM2 och proceduren OUTSYM. Här används alla tre pekarna och SYMLIST till att tala om vilka instruktioner som kan skrivas på utfilen. OUTSYM är det s.k. snittet som gör utmatningen. OUTSYM tar också reda på om något argument ska matas ut eller inte. Den har också kontroll över listan som innehåller kommentarerna och matar ut dessa då den instruktion som matas ut är en kommentarsymbol.

5.5 Slutord om optimeringen

Här ska framföras några synpunkter på optimeringsdelen av vårt projekt och även behandla inriktningen på vårt arbete. Vi kommer också att visa en kodsekvens som borde optimeras bort, men som går utanför ramen på projektet.

Det skall framhållas att vår huvudsakliga uppgift var att konstruera en kodgenerator med hyfsade krav på mängden kod den fick generera. Uppgiften var alltså inte att prestera minimal PBS-kod för motsvarande Pascal-program. Vi är nöjda med det resultat i form av PBS-kod vi får från PBSGEN + PBSOPT och om det görs en jämförelse mellan det PBS-program vi genererat och motsvarande handkodade PBS-program så visar det sig att mängden kod är likvärdig (Avesta-programmet i kapitel 7).

PBSOPT konstruerades till en början enkom för att ta bort redundanta kodsekvenser, emellertid kom PBSOPT att få fler uppgifter senare. Vi är medvetna om att med utökad satsning på optimering så skulle man kunna reducera mängden maskingenererad kod med ytterligare 10-15%. Att procenttalet varierar och att det antagligen borde ha ett ännu större spel beror på hur det Pascal-program ser ut som koden ska genereras från. Olika sats-konstruktioner kan nämligen optimeras olika mycket.

Ett exempel på redundant kod som vi inte ängnat någon tid åt att försöka optimera bort, finns i fig 5.10, Pascal-koden för den PBS-koden finns i samma figur. Att koden överhuvudtaget kommer till beror på att vi endast har ett stegs look-ahead i PBSGEN:s hantering av if-satser. Exemplet är ett specialfall och om man studerar den kod som maskingenereras så kommer man att upptäcka många sådana här specialfall som skulle kunna optimeras till bättre kod. Just det att det är ett specialfall gör att programkoden som krävs för att ta hand om detsamma kan lätt komma att innehålla rätt komplexa moment. Därmed blir tidsinsatsen för att lösa problemet rätt stor och vi har tidigare framhållit att det inte är vår uppgift just nu, men i ett annat projekt skulle man kunna ägna sig helt åt att prestera minimal PBS-kod, dvs göra en så god optimering som möjligt.

```
Pascal-kod                                  PBS-kod

...                                         ...
procedure CALL;                             AD      A
begin                                       SE   1004
   if D then ... ;                          RS
   if E then ... ;                          AD      B
end(* CALL *);                              SE   1005
                                            RS
if A then                                   AD      C
begin                                       SE   1006
   if B then                                RS
   begin                                    AD      D
      if C then                             SE   1007
      begin                                 ...
         CALL;                              RS
         if F then                         -AD   1004
...                                        !AD   1005
                    Onödig kod   =>        !AD   1006
                                           !SSN
                                           -RS
                                            AD      F
                                            SE   1007
                                            ...
```

figur 5.10
Illustrerar en kodsekvens vi inte optimerar bort

Det motiveras också av att den förbättring av kodmängden som optimeringen av ett speciallfall ger, är rätt så liten. Egentligen ska man betrakta den förbättring som optimeringen av alla specialfall ger. Att då ägna sig åt ett i mängden och inte ta hand om de andra specialfallen, känns inte meningsfullt.

6 ERFARENHETER

Vi har under arbetet med projektet funderat på vilka möjligheter m m ett högnivåspråk för PC-system erbjuder. Under projektets gång har vi kommit fram till att ett sådant språk behöver kraftfulla primitiver ej speciellt anpassade till PC-system, utan av generell karaktär. Språket måste också innehålla primitiver för att generellt kunna nå maskinvaran, t ex vid knytning av specifika adresser till speciella variabler.

Vi har funnit många av Pascals egenskaper lämpliga för ändamålet. Dock har P-koden som mellansteg satt en del begränsningar som vi har varit tvungna "konstruera" bort i kodgeneratorn. Ett par av nackdelarna med P-koden har varit följande :

- Ej möjligt att syntax-kontrollera det Pascal-program som kommer som indata i form av P-kod med hänsyn till den delmängd som används för PC-programmering.

- Ej möjligt att föra över variabelnamn i P-koden dvs svårt att enkelt binda fysisk adress till variabel

- Begreppen i Pascal svarar inte alltid mot vad man vill göra i PC-systemet

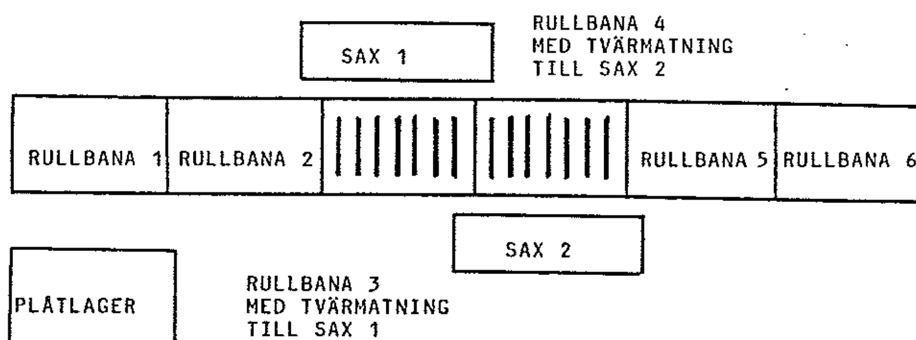
Trots att mycket är vunnet med att programmera PC-system i Pascal jämfört med PBS-kod är Pascal fortfarande, precis som PBS-kod, ett sekventiellt språk. Eftersom många händelser i den verklighet som ska styras av PC-systemet sker parallellt skulle det vara lämpligt att programmera i ett språk som är utrustat med någon form av beskrivning av parallella aktiviteter. Det skulle således vara möjligt att definiera aktiviteter som skall exekveras skenbart parallellt i PC-systemet.

För att få ett enkelt men kraftfullt språk med egenskaper lämpliga för PC-system torde man få specialdesigna ett språk. Språket skulle, som vi ser det, vara ganska likt den delmängd av Pascal vi använt, dock med utvidgningar för beskrivning av parallella aktiviteter och kanske primitiver för sekvensstyrning. Att generera kod över en mellankod är lämpligt då man på så sätt lättare kan generera kod för många olika typer av PC-system. Man skulle även genom att välja en lämplig utformning på mellankoden föra över "lagom" mycket information till kodgenereringen.

7 PROGRAMEXEMPEL

7.1 Avesta jernverk

Detta exempel beskriver en existerande anläggning på Avesta Jernverk. Anlägningen används för att klippa rent kanterna på kallvalsade plåtar. Se figur nedan för en översiktsbild på anläggningen.



Av bilden framgår att anläggningen består av sex rullbanor, två saxar och ett plåtlager. Två av rullbanorna är dessutom försedda med tvärmatning till saxarna. Alla banor är försedda med matning framåt. En av tvärbanorna kan också drivas bakåt. Alla funktioner som PBS-mini ska styra eller känna av är utrustade med lämpliga givare och kontaktdon.

Behandlingen av en plåt tillgår på följande sätt:

1. Från plåtlager till rullbana 1 (Styrs ej av PBS-minin)

2. Från rullbana 1 till rullbana 2 där den mäts och klassas

3. Till rullbana 3 där vänsterkanten klipps enligt klassningen

4. Till rullbana 4 där klippes högerkanten med ett enkelklipp oavsett klassning

5. Rullbana 5 och 6 är transportsträckor till utrustning som inte styrs av PBS-minin

Under punkt 2 nämndes klassning och mätning. Klassning innebär vilken typ av klippning som skall utföras i steg 3: enkelklipp, dubbelklipp eller skrot. Skrot innebär inget klipp alls. Anledningen att införa automatik var att kunna utföra så mycket som möjligt parallellt och dessutom öka säkerheten för personalen.

Av exemplet framgår klart möjligheten att strukturera ett problem med hjälp av datastrukturer och if-satser. Procedurer kommer inte till så stor användning här p g a att problemet inte innehåller flera händelser av lika slag.

```
program avesta(prr);

const numofedge = 25;
      numoftimers = 13;

type

      oprec = record
         klar,reset,
         dubbel,skrot,
         lampa1,lampa2: boolean
      end (* oprec *);

      plaatrec = record
         laang,mellan,kort,          (* laengd paa plaaten *)
         dubbel,enkel,skrot:boolean (* typ av bearbetning *)
      end (* plaatrec *);

      rbanarec = record
         motor: boolean;  (* driftsignal till motor *)
         plaat: plaatrec;  (* plaat f.n. paa banan *)
         slut: boolean;
         case nr: integer of
            1,5,6: ( );
            2: (start2,mitt2,utmatningsfas: boolean);
            3: (mitt3,plaatpaabana,stopp3:boolean;
                plaatut:plaatrec);
            4: (uppe,nere,hoej,saenk,
                stopp4,flankslut4: boolean);
      end (* rbanarec *);

      saxrec = record
         fram,back,hoej,nere,uppe,
         klipp,klippklar: boolean;
         case nr: integer of
         1: (anslag: array[1..3] of boolean;
             ute1,ute2: boolean;
             klipp1klart,klipp1flank,klipp2klart,
             klippt,motanslag,
             flanknere: boolean);
         2: (mhake13hoej,mhake2hoej,
             mhakarsaenk,
             mhake13uppe,mhake2uppe,
             mhakarnere,klippredo,
             klartklipp,mothakar: boolean);
      end;

var

      rullb: array[1..6] of rbanarec;
      operatoer: oprec;
      sax: array[1..2] of saxrec;
      mskrot,mdubb: boolean;
      fedgeklar,redgeklar: boolean;
      skift23: boolean;
      registrering: boolean;

      e: array[1..numofedge] of boolean;  (* edge-celler *)
      puls1hz,puls10hz: boolean;
      timerut: array[1..numoftimers] of boolean;
      timerself: array[1..numoftimers] of boolean;
```

```
        bannr,saxnr,i: integer;

procedure timer(inuttr,holduttr: boolean; var tidbas,
                      utcell:boolean;tid: integer); external;
procedure clearpbs(villkor: boolean); external;
function edge(uttr: boolean;var cell: boolean): boolean;
                                                  external;
function exor(uttr: boolean;var cell: boolean): boolean;
                                                  external;
procedure origin(var boolvar: boolean; oktadr: integer);
                                                  external;

procedure drift(var cell: boolean; startvillkor,
                                 stoppvillkor: boolean);
begin
   if startvillkor then cell:= true;
   if stoppvillkor then cell:= false;
end;

begin  (* huvudprogrammet *)

   for bannr:=1 to 6 do rullb[bannr].nr:= bannr;
   for saxnr:=1 to 2 do  sax[saxnr].nr:=saxnr;

   (* origin-deklarationer foer in- och ut-gaangar: *)

   with operatoer do
   begin
      origin(reset,400);origin(klar,425);origin(dubbel,426);
      origin(skrot,427);origin(lampa1,16);origin(lampa2,17);
   end;
   for bannr:=1 to 6 do
   with rullb[bannr] do
   begin
      origin(slut,400+bannr); origin(motor,bannr);
      if bannr=2 then
         begin origin(start2,411); origin(mitt2,412); end;
      if bannr=3 then origin(mitt3,413);
      if bannr=4 then
      begin
         origin(uppe,414); origin(nere,415);
         origin(hoej,14); origin(saenk,15);
      end;
   end;
   with sax[1] do
   begin
      origin(fram,12); origin(back,7);
      origin(hoej,10); origin(klipp,11);
      origin(nere,423); origin(klippklar,424);
      origin(ute1,416); origin(ute2,417);
      for i:=1 to 3 do origin(anslag[i],419+i);
   end;
   with sax[2] do
   begin
      origin(back,40); origin(hoej,41);
      origin(klipp,42); origin(mhake13hoej,43);
      origin(mhake2hoej,44); origin(mhakarsaenk,45);
      origin(nere,430); origin(klippklar,432);
      origin(uppe,431); origin(mhakarnere,435);
```

```
   origin(mhake13uppe,436); origin(mhake2uppe,437);
   origin(klippredo,433);
end;
origin(puls1hz,1001);
origin(puls10hz,1000);


writeln(prr,'operatoer');          (* kod foer operatoer *)
with operatoer do
begin
   clearpbs(reset);
   fedgeklar:= edge(klar,e[4]);
   redgeklar:= edge(not klar,e[5]);
   mdubb:= edge(dubbel,e[1]) or mdubb and dubbel and not
             redgeklar;
   mskrot:= edge(skrot,e[2]) or mskrot and skrot and not
             redgeklar;
end;

with rullb[1].plaat do        (* initiering av ny plaat *)
begin
   kort:=false; mellan:=false; laang:=false;
   enkel:=false; dubbel:=false; skrot:=false;
end;

writeln(prr,'rullbana 1:');      (* kod foer rullbana 1 *)
timer(rullb[1].slut,false,puls1hz,timerut[1],3);
drift(rullb[1].motor,
      timerut[1] and not rullb[2].slut,
      rullb[2].slut);

                          (* skift av plaat bana 1 -> 2 *)
if edge(rullb[2].slut,e[3]) then
   rullb[2].plaat:=rullb[1].plaat;

writeln(prr,'rullbana 2:');      (* kod foer rullbana 2 *)
with rullb[2] do
begin
   with plaat do
   begin     (* registrering av laengder & bearbetning *)
      registrering:= slut and not motor and fedgeklar
                      and not utmatningsfas;
      if registrering then
      begin
         if not mitt2 then kort:=true;
         if mitt2 and not start2 then mellan:=true;
         if start2 then laang:=true;

         if not exor(mdubb,mskrot) then enkel:=true;
         if not mdubb and mskrot then skrot:=true;
         if mdubb and not mskrot then dubbel:=true;
      end;
   end;
   drift(utmatningsfas,
         registrering,
         exor(utmatningsfas,slut));
   drift(motor,
         rullb[1].slut and not slut or utmatningsfas and
         (not rullb[3].plaatpaabana and
          sax[1].nere or timerut[2]),
         exor(utmatningsfas,slut));
end;
                             (* skift av plaat bana 2 -> 3 *)
skift23:=edge(rullb[3].mitt3,e[6]);
if skift23 then rullb[3].plaat:= rullb[2].plaat;

writeln(prr,'rullbana 3:');     (* kod foer rullbana 3 *)
with rullb[3] do
begin
   timer(motor,false,puls1hz,timerut[2],3);
   drift(plaatpaabana,
         edge(mitt3,e[17]),
         edge(motor,e[7]));
   stopp3:= edge(not motor,e[8]);
   if stopp3 and plaatpaabana then plaatut:= plaat;
   drift(timerself[4],
         plaat.mellan and skift23,
         timerut[4]);
   timer(timerself[4],false,puls10hz,timerut[4],16);
   drift(timerself[5],
         (plaat.laang or plaat.kort) and edge(slut,e[9]),
         timerut[5]);
   timer(timerself[5],false,puls10hz,timerut[5],13);
   drift(motor,
        not plaatpaabana and sax[1].nere and
        rullb[2].utmatningsfas or
        plaatpaabana and (sax[1].ute1 or sax[1].ute2)
        and sax[1].flanknere and sax[2].nere
        and not rullb[4].slut and
        rullb[4].uppe and sax[2].mhakarnere,
        not plaatpaabana and edge(rullb[4].slut,e[20]) or
        timerut[4] or timerut[5]);
end;
                             (* skift av plaat bana 3 -> 4 *)
if edge(not rullb[3].plaatpaabana,e[12])
then rullb[4].plaat:=rullb[3].plaatut;

writeln(prr,'sax 1:');                (* kod foer sax 1 *)
with sax[1] do
if rullb[3].plaatpaabana then
begin
   klipp1flank:=edge(klipp1klart,e[13]);
   flanknere:=edge(nere,e[21]);
   if rullb[3].plaat.enkel and not klipp1klart or
      rullb[3].plaat.dubbel and not klipp2klart then
   begin
      motanslag:=anslag[1] and anslag[2] or
                 anslag[1] and anslag[3] or
                 anslag[2] and anslag[3];
      operatoer.lampa1:=motanslag;
      drift(hoej,
            rullb[3].stopp3 or klipp1flank,
            motanslag);
      timer(hoej,false,puls1hz,timerut[6],2);
      fram:= timerut[6];
      drift(timerself[7],
            flanknere,
            timerut[7]);
      timer(timerself[7],false,puls1hz,timerut[7],1);
      klipp:=timerself[7];
      klippt:=edge(klippklart,e[22]);
      if klippt and klipp1klart then klipp2klart:= true;
```

```
      if klippt then klipp1klart:= true;
   end
   else
   begin
      drift(hoej,
            rullb[3].stopp3 or klipp1flank or
            edge(klipp2klart,e[14]),timerut[8]);
      timer(ute1 or ute2,false,puls10hz,timerut[8],10);
      timer(hoej,false,puls10hz,timerut[9],20);
      back:= timerut[9];
   end
end
else
begin
   klipp1klart:= false;
   klipp2klart:= false;
end;

writeln(prr,'rullbana 4:');     (* kod foer rullbana 4 *)
with rullb[4] do
begin
   flankslut4:=edge(slut,e[24]);
   drift(timerself[12],
         flankslut4 and plaat.kort,
         timerut[12]);
   timer(timerself[12],false,puls10hz,timerut[12],27);
   drift(timerself[13],
         flankslut4 and plaat.laang,
         timerut[13]);
   timer(timerself[13],false,puls10hz,timerut[13],40);
   drift(motor,
      sax[2].klartklipp and not rullb[5].slut or
      rullb[3].plaatpaabana and not slut and uppe and
      sax[2].nere and sax[2].mhakarnere and
      sax[1].flanknere and (sax[1].ute1 or sax[1].ute2),
      timerut[12] or timerut[13] or plaat.mellan and
      flankslut4 or not rullb[3].plaatpaabana and
      edge(not slut,e[25]));
   drift(hoej,
      not uppe and (sax[2].klartklipp or
      rullb[3].plaatpaabana and not slut),
      uppe);
   stopp4:=edge(not motor,e[18]);
end(* with *);

writeln(prr,'sax 2:');              (* kod foer sax 2 *)
with sax[2] do
if rullb[4].slut
then
   if not klartklipp then
   begin
      with rullb[4] do drift(saenk,stopp4,nere);
      if not rullb[4].plaat.skrot then
      begin
         drift(hoej,
               rullb[4].stopp4,
               mothakar);
         with rullb[4] do
         if stopp4 then
         begin
            mhake13hoej:=true;
            if plaat.kort or plaat.mellan
            then mhake2hoej:=true;
         end(* with *);
         if klippredo then mothakar:=true;
         drift(back,
               uppe and rullb[4].nere and mhake13uppe and
               ((rullb[4].plaat.kort or
                 rullb[4].plaat.mellan) and
               mhake2uppe or rullb[4].plaat.laang and
               not mhake2uppe), mothakar);
         drift(timerself[10],
               mothakar and rullb[4].nere and
               edge(nere,e[19]),
               timerut[10]);
         timer(timerself[10],false,puls10hz,
               timerut[10],10);
         klipp:=timerself[10];
         if klippklart then klartklipp:=true;
      end
      else
      begin
         drift(timerself[11],
               rullb[4].stopp4,
               timerut[11]);
         timer(timerself[11],false,puls10hz,
               timerut[11],10);
         klipp:=timerself[11];
         if klippklart then klartklipp:=true;
      end;
   end
   else
   begin
      mhake13hoej:=false;
      mhake2hoej:=false;
      mothakar:=false;
      mhakarsaenk:=not mhakarnere;
   end
else klartklipp:=false;

writeln(prr,'rullbana 5:');     (* kod foer rullbana 5 *)
drift(rullb[5].motor,
      rullb[5].slut and not rullb[6].slut or
      sax[2].klartklipp and not rullb[5].slut,
      edge(rullb[5].slut,e[15]) or not sax[2].klippklart
      and edge(not rullb[5].slut,e[23]));

writeln(prr,'rullbana 6:');     (* kod foer rullbana 6 *)
drift(rullb[6].motor,
      rullb[5].slut and not rullb[6].slut,
      edge(rullb[6].slut,e[16]));
operatoer.lampa2:= rullb[6].slut;
end.
```

```
*C* OPERATOER
 AD          400
*CP
 AD          425
 EDGE       1020
*SE         1021
 ADN         425
 EDGE       1023
*SE         1024
 AD          426
 EDGE       1026
*SE         1025
 AD         1027
 AD          426
 ADN        1024
 OR         1025
*SE         1027
 AD          427
 EDGE       1031
*SE         1030
 AD         1032
 AD          427
 ADN        1024
 OR         1030
*SE         1032
*SEN        1033
*SEN        1034
*SEN        1035
*SEN        1036
*SEN        1037
*SEN        1040
*C* RULLBANA 1:
 ADN         401
 OR         1001
 EDGE       1041
 CNT+       1042
 CNT+       1043
*COT+       1044
 AD         1044
 ADN         402
*SE         1045
 AD          402
*SE         1046
 AD         1045
*SSR           1
 AD         1046
*RSR           1
 AD          402
 EDGE       1047
*SSN
 AD         1040
*SE         1050
 AD         1036
*SE         1051
 AD         1037
*SE         1052
 AD         1033
*SE         1053
 AD         1034
*SE         1054
 AD         1035
```

```
*SE         1055
*RS
*C* RULLBANA 2:
 AD          402
 ADN           2
 AD         1021
 ADN        1056
*SE         1057
 AD         1057
*SE         1004
 ADN         412
*SE         1005
 AD         1004
 AD         1005
*SSR        1053
 AD          412
 ADN         411
*SE         1005
 AD         1004
 AD         1005
*SSR        1054
 AD          411
*SE         1005
 AD         1004
 AD         1005
*SSR        1055
 AD         1027
 EXN        1032
*SE         1005
 AD         1004
 AD         1005
*SSR        1051
 ADN        1027
 AD         1032
*SE         1005
 AD         1004
 AD         1005
*SSR        1050
 AD         1027
 ADN        1032
*SE         1005
 AD         1004
 AD         1005
*SSR        1052
 AD         1056
 EX          402
*SE         1060
 AD         1057
*SE         1061
 AD         1060
*SE         1062
 AD         1061
*SSR        1056
 AD         1062
*RSR        1056
 AD         1056
 EX          402
*SE         1066
 ADN        1063
 AD          423
 OR         1064
```

```
*SE         1065
 AD         1065
 AD         1056
 OR          401
 ADN         402
*SE         1067
 AD         1066
*SE         1070
 AD         1067
*SSR           2
 AD         1070
*RSR           2
 AD          413
 EDGE       1071
*SE         1072
 AD         1072
*SSN
 AD         1050
*SE         1073
 AD         1051
*SE         1074
 AD         1052
*SE         1075
 AD         1053
*SE         1076
 AD         1054
*SE         1077
 AD         1055
*SE         1100
*RS
*C* RULLBANA 3:
 ADN           3
 OR         1001
 EDGE       1101
 CNT+       1102
 CNT+       1103
*COT+       1064
 AD          413
 EDGE       1105
*SE         1104
 AD            3
 EDGE       1107
*SE         1106
 AD         1104
*SE         1110
 AD         1106
*SE         1111
 AD         1110
*SSR        1063
 AD         1111
*RSR        1063
 ADN           3
 EDGE       1112
*SE         1113
 AD         1113
 AD         1063
*SSN
 AD         1073
*SE         1114
 AD         1074
*SE         1115
```

```
 AD         1075
*SE         1116
 AD         1076
*SE         1117
 AD         1077
*SE         1120
 AD         1100
*SE         1121
*RS
 AD         1077
 AD         1072
*SE         1123
 AD         1122
*SE         1124
 AD         1123
*SSR        1125
 AD         1124
*RSR        1125
 ADN        1125
 OR         1000
 EDGE       1126
 CNT-       1127
 CNT-       1130
 CNT-       1131
 CNT-       1132
 CNT+       1133
*COT+       1122
 AD          403
 EDGE       1135
*SE         1134
 AD         1100
 OR         1076
*SE         1136
 AD         1136
 AD         1134
*SE         1140
 AD         1137
*SE         1141
 AD         1140
*SSR        1142
 AD         1141
*RSR        1142
 ADN        1142
 OR         1000
 EDGE       1143
 CNT+       1144
 CNT-       1145
 CNT+       1146
 CNT+       1147
*COT+       1137
 AD          404
 EDGE       1153
*SE         1152
 AD          416
 OR          417
*SE         1150
 AD         1150
 AD         1063
 AD         1151
 AD          430
 ADN         404
```

```
 AD        414
 AD        435
 ORN      1063
 AD        423
 AD       1056
*SE       1154
 ADN      1063
 AD       1152
 OR       1122
 OR       1137
*SE       1155
 AD       1154
*SSR         3
 AD       1155
*RSR         3
 ADN      1063
 EDGE     1156
*SSN
 AD       1114
*SE       1157
 AD       1115
*SE       1160
 AD       1116
*SE       1161
 AD       1117
*SE       1162
 AD       1120
*SE       1163
 AD       1121
*SE       1164
*RS
*C* SAX 1:
 AD       1063
*SE       1004
 AD       1004
*SSN
 AD       1165
 EDGE     1167
*SE       1170
 AD        423
 EDGE     1172
*SE       1151
*RS
 AD       1074
 ADN      1165
 OR       1075
 ADN      1173
*SE       1005
 AD       1004
 AD       1005
*SSN
 AD        420
 AD        421
 OR        420
 AD        422
 OR        421
 AD        422
*SE       1174
 AD       1174
*SE         16
 AD       1113
```

```
 OR       1170
*SE       1175
 AD       1174
*SE       1176
*RS
 AD       1175
*SE       1006
 AD       1004
 AD       1005
 AD       1006
*SSR        10
 AD       1176
*SE       1006
 AD       1004
 AD       1005
 AD       1006
*RSR        10
 AD       1004
 AD       1005
*SSN
 ADN        10
 OR       1001
 EDGE     1177
 CNT-     1200
 CNT+     1201
*COT+     1202
 AD       1202
*SE         12
 AD       1151
*SE       1204
 AD       1203
*SE       1205
*RS
 AD       1204
*SE       1006
 AD       1004
 AD       1005
 AD       1006
*SSR      1206
 AD       1205
*SE       1006
 AD       1004
 AD       1005
 AD       1006
*RSR      1206
 AD       1004
 AD       1005
*SSN
 ADN      1206
 OR       1001
 EDGE     1207
 CNT+     1210
*COT+     1203
 AD       1206
*SE         11
 AD        424
 EDGE     1212
*SE       1213
*RS
 AD       1213
 AD       1165
```

```
*SE       1006
 AD       1004
 AD       1005
 AD       1006
*SSR      1173
 AD       1213
*SE       1006
 AD       1004
 AD       1005
 AD       1006
*SSR      1165
 AD       1004
 ADN      1005
*SSN
 AD       1173
 EDGE     1215
*SE       1214
 AD       1113
 OR       1170
 OR       1214
*SE       1217
 AD       1216
*SE       1220
*RS
 AD       1217
*SE       1006
 AD       1004
 ADN      1005
 AD       1006
*SSR        10
 AD       1220
*SE       1006
 AD       1004
 ADN      1005
 AD       1006
*RSR        10
 AD       1004
 ADN      1005
*SSN
 AD        416
 OR        417
*SE       1220
 ADN      1220
 OR       1000
 EDGE     1221
 CNT-     1222
 CNT+     1223
 CNT-     1224
 CNT+     1225
*COT+     1216
 ADN        10
 OR       1000
 EDGE     1226
 CNT-     1227
 CNT-     1230
 CNT+     1231
 CNT-     1232
 CNT+     1233
*COT+     1234
 AD       1234
*SE          7
```

```
*RS
 ADN      1004
*SSN
*SEN      1165
*SEN      1173
*RS
*C* RULLBANA 4:
 AD        404
 EDGE     1236
*SE       1237
 AD       1237
 AD       1162
*SE       1241
 AD       1240
*SE       1242
 AD       1241
*SSR      1243
 AD       1242
*RSR      1243
 ADN      1243
 OR       1000
 EDGE     1244
 CNT+     1245
 CNT+     1246
 CNT-     1247
 CNT+     1250
 CNT+     1251
*COT+     1240
 AD       1237
 AD       1164
*SE       1253
 AD       1252
*SE       1254
 AD       1253
*SSR      1255
 AD       1254
*RSR      1255
 ADN      1255
 OR       1000
 EDGE     1256
 CNT-     1257
 CNT-     1260
 CNT-     1261
 CNT+     1262
 CNT-     1263
 CNT+     1264
*COT+     1252
 ADN       404
 EDGE     1270
*SE       1267
 AD        416
 OR        417
*SE       1266
 AD       1266
 AD       1063
 ADN       404
 AD        414
 AD        430
 AD        435
 AD       1151
 OR       1265
```

```
 ADN     405
*SE     1271
 AD     1240
 OR     1252
 OR     1163
 AD     1237
 ORN    1063
 AD     1267
*SE     1272
 AD     1271
*SSR       4
 AD     1272
*RSR       4
 AD     1063
 ADN     404
 OR     1265
*SE     1272
 AD     1272
 ADN     414
*SE     1273
 AD      414
*SE     1274
 AD     1273
*SSR      14
 AD     1274
*RSR      14
 ADN       4
 EDGE   1275
*SE     1276
*C* SAX 2:
 AD      404
*SE     1004
 ADN    1265
*SE     1005
 AD     1004
 AD     1005
*SSN
 AD     1276
*SE     1277
 AD      415
*SE     1300
*RS
 AD     1277
*SE     1006
 AD     1004
 AD     1005
 AD     1006
*SSR      15
 AD     1300
*SE     1006
 AD     1004
 AD     1005
 AD     1006
*RSR      15
 AD     1004
 AD     1005
*SSN
*RS
 ADN    1157
*SE     1006
 AD     1004

 AD     1005
 AD     1006
*SSN
 AD     1276
*SE     1302
 AD     1301
*SE     1303
*RS
 AD     1302
*SE     1007
 AD     1004
 AD     1005
 AD     1006
 AD     1007
*SSR      41
 AD     1303
*SE     1007
 AD     1004
 AD     1005
 AD     1006
 AD     1007
*RSR      41
 AD     1004
 AD     1005
 AD     1006
*SSN
*RS
 AD     1276
*SE     1007
 AD     1004
 AD     1005
 AD     1006
 AD     1007
*SSR      43
 AD     1162
 OR     1163
*SE     1010
 AD     1004
 AD     1005
 AD     1006
 AD     1007
 AD     1010
*SSR      44
 AD      433
*SE     1007
 AD     1004
 AD     1005
 AD     1006
 AD     1007
*SSR    1301
 AD     1004
 AD     1005
 AD     1006
*SSN
 AD     1162
 OR     1163
*SE     1303
 AD     1303
 AD      437
 OR     1164
 ADN     437

*SE     1304
 AD     1304
 AD      431
 AD      415
 AD      436
*SE     1305
 AD     1301
*SE     1306
*RS
 AD     1305
*SE     1007
 AD     1004
 AD     1005
 AD     1006
 AD     1007
*SSR      40
 AD     1306
*SE     1007
 AD     1004
 AD     1005
 AD     1006
 AD     1007
*RSR      40
 AD     1004
 AD     1005
 AD     1006
*SSN
 AD      430
 EDGE   1307
*SE     1306
 AD     1301
 AD      415
 AD     1306
*SE     1311
 AD     1310
*SE     1312
*RS
 AD     1311
*SE     1007
 AD     1004
 AD     1005
 AD     1006
 AD     1007
*SSR    1313
 AD     1312
*SE     1007
 AD     1004
 AD     1005
 AD     1006
 AD     1007
*RSR    1313
 AD     1004
 AD     1005
 AD     1006
*SSN
 ADN    1313
 OR     1000
 EDGE   1314
 CNT-   1315
 CNT+   1316
 CNT-   1317

 CNT+   1320
*COT+   1310
 AD     1313
*SE       42
*RS
 AD      432
*SE     1007
 AD     1004
 AD     1005
 AD     1006
 AD     1007
*SSR    1265
 AD     1004
 AD     1005
 ADN    1006
*SSN
 AD     1276
*SE     1322
 AD     1321
*SE     1323
*RS
 AD     1322
*SE     1007
 AD     1004
 AD     1005
 ADN    1006
 AD     1007
*SSR    1324
 AD     1323
*SE     1007
 AD     1004
 AD     1005
 ADN    1006
 AD     1007
*RSR    1324
 AD     1004
 AD     1005
 ADN    1006
*SSN
 ADN    1324
 OR     1000
 EDGE   1325
 CNT-   1326
 CNT+   1327
 CNT-   1330
 CNT+   1331
*COT+   1321
 AD     1324
*SE       42
*RS
 AD      432
*SE     1007
 AD     1004
 AD     1005
 ADN    1006
 AD     1007
*SSR    1265
 AD     1004
 ADN    1005
*SSN
*SEN      43
```

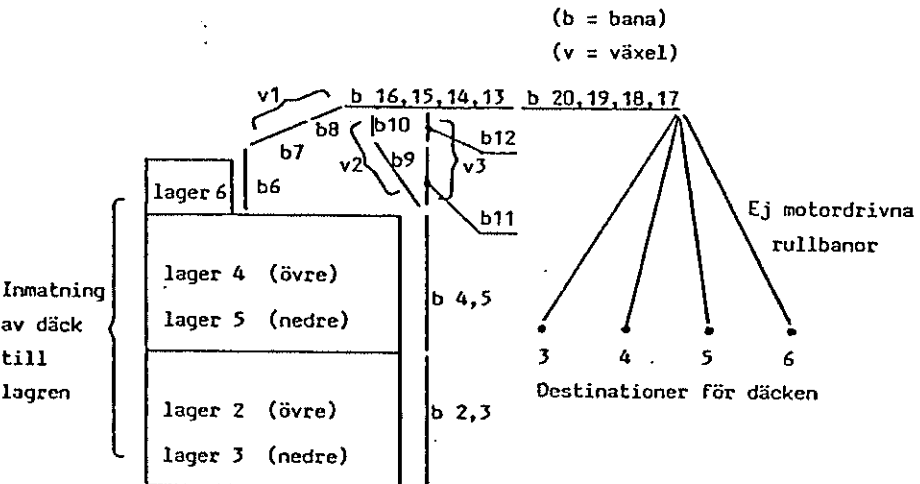
```
*SEN         44
*SEN       1301
 ADN        435
*SE          45
*RS
 ADN       1004
*RSR       1265
*C* RULLBANA 5:
 AD         405
 EDGE      1333
*SE        1332
 ADN        405
 EDGE      1335
*SE        1334
 AD         405
 ADN        406
 OR        1265
 ADN        405
*SE        1336
 ADN        432
 AD        1334
 OR        1332
*SE        1337
 AD        1336
*SSR          5
 AD        1337
*RSR          5
*C* RULLBANA 6:
 AD         406
 EDGE      1340
*SE        1337
 AD         405
 ADN        406
*SE        1341
 AD        1337
*SE        1342
 AD        1341
*SSR          6
 AD        1342
*RSR          6
 AD         406
*SE          17
 END
```

7.2 Gislaved däcklager

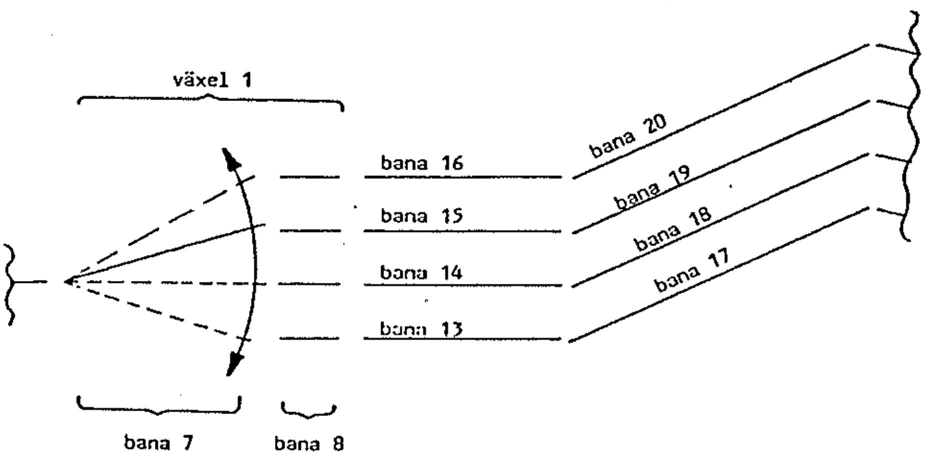
Det här exemplet beskriver, även det, en existerande anläggning: ett däcklager hos Gislaveds däcksfabrik. Däcken lagras stående i rännor och kan rullas ut ur dessa rännor genom att ett par dörrar i ena änden av rännan öppnas. Lagret är indelat i fem mindre sektioner: fyra med femton rännor vardera och en med sex rännor. Lagret avses fungera som buffert till fyra däcktestmaskiner. Dessa maskiner ska gå så kontinuerligt som möjligt, dvs inga avbrott då det finns däck som kan testas. För att försörja maskinerna med däck, har det byggts upp ett nätverk av rullbanor. Då alla maskiner ska kunna få däck från alla rännor så måste många av banorna vara gemensamma i nätverket. Uppgiften är alltså att styra anläggningen så optimalt som möjligt.

En total styrning av anläggningen kräver en dator för att uppnå optimal styrning och administration. Underordnat denna dator finns en PBS som sköter start och stopp av sekvenssteg, banor m m. Då lagret innehåller fem sektioner (kallas härefter för lager) och fyra destinationer så måste det finnas tjugo vägar genom nätverket av banor. Interfacet mellan PBS och dator består av just dessa tjugo vägar, datorn sätter de celler i PBS:n som motsvarar de vägar datorn vill ha uppkopplade. I programmet motsvaras detta av array:en storetodest. Det går endast att ha tre vägar uppkopplade samtidigt, trots att det vore mest optimalt att kunna ha fyra vägar samtidigt. Begränsningen sätts av att det endast finns tre växlar och varje väg utnyttjar en växel. Figuren på nästa sida är ett försök att förklara de olika vägarna i anläggningen, tyvärr har det inte gått att göra någon bild av anläggningen. Att det är en komplex anläggning att styra framgår av tabellerna i figuren.

ÖVERSIKTSBILD PÅ GISLAVEDS DÄCKLAGER



ÖVERSIKTSBILD PÅ VÄXEL 1 OCH EFTERFÖLJANDE BANOR



till destination	banor före destinationen
6	20 och 16
5	19 och 15
4	18 och 14
3	17 och 13

2	3	2	3	1	via växel
1	5	9	13	17	vägarna som
2	6	10	14	18	däck kan
3	7	11	15	19	transporteras
4	8	12	16	20	genom anl.
2	3	4	5	6	från lager

Talen 1 till 20 i tabellen ovan mosvaras alltså av array:en storetodest[] i Pascal-programmet. Array:erna path[] innehåller alltså också dessa tal för att uppnå en mer flexibel adressering.

växel	består av banorna
1	7 och 8
2	8 och 10
3	11 och 12

Banan framför varje lager har samma nr som lagret.

emptysign	växel	index till storetrackseq[]	index till tracknr[] 1	2	3	4	lagernamn
1	2	1	2	4	9	10	st2and4
2	3	2	3	5	11	12	st3and5
3	1	3	–	6	7	8	st6

Med emptysign som index kan man signalera till datorn att en bansekvens är tom, här är det den som beskrivs av array:en storetrackseq[] och proceduren starting. Även växlarna påverkas av detta emptysign. Av tabellen framgår att ett visst lager alltid är bundet till samma växel, det går alltså inte att skicka däck från lager 2 genom växel 3 o s v. Tracknr[1] används ej för lager 6, då där ej existerar någon bana det kan svara mot.

Hela anläggningen styrs i sekvens med följande steg:

1. Starta banorna närmast destinationen
2. Ställ in växeln och starta banorna i den
3. Starta banorna framför lagret
4. Öppna dörrarna till rännorna och släpp ut däck på banorna

Misslyckas starten av ett av stegen så kopplas den påbörjade sekvensen ner.

I Pascal-programmet har vi representerat varje sekvenssteg med en procedure och varje steg har även sina egna datastrukturer i form av records m m. En hel del avancerad programmering används i exemplet t ex indirektadresseringen med hjälp av array:erna path och tracknr m m. Path används för att ange vilka vägar genom systemet som påverkar just det sekvenssteget.

```
program Gislaved(prr);

const nrofdoors = 15;
      nrofcerrs = 4;
      switchlevels = 4;
      trksinfirstseq = 4;
      trksinsecondseq = 2;

type  trackrange = 2..20;
      pathrange = 1..20;
      switchrange = 1..3;
      storerange = 2..6;
      emptyrange = 1..3;

      doorrec = record
         signal,opendoor : boolean
      end(* doorrec *);

      storename = (str2,str3,str4,str5,str6);

      storerec = record
         name : storename;
         trackinfrontofstore : trackrange;
         door : array [1..nrofdoors] of doorrec;
         wait,open,close,error,counter,emem : boolean;
         cerr : array [1..nrofcerrs] of boolean;
         path : array [1..4] of pathrange;
         timeout1,timeout2 : boolean
      end(* storerec *);

      pistonrec = record
         push,pull,pushed,pulled : boolean
      end(* pistonrec *);

      pistontype = (short,long);

      switchname = (sw1,sw2,sw3);

      switchrec = record
         name : switchname;
         emptysign : integer;
         piston : array [pistontype] of pistonrec;
         path : array [1..8] of pathrange;
         level : array [1..switchlevels] of boolean;
         error,wait,timeout : boolean
      end(* switchrec *);

      trackrec = record
         engine,verify : boolean
      end(* trackrec *);

      namefirstseq = (st2and4,st3and5,st6);

      firsttrackseqrec = record
         name : namefirstseq;
         emptysign : integer;
         switchnr : switchrange;
         path : array [1..8] of pathrange;
         tracknr : array[1..trksinfirstseq] of trackrange;
         wait,timeout,error : boolean
      end(* firsttrackseqrec *);

      namesecondseq = (dest3,dest4,dest5,dest6);

      secondtrackseqrec = record
         name : namesecondseq;
         path : array [1..5] of pathrange;
         tracknr : array[1..trksinsecondseq] of trackrange;
         error,wait,destfull,photocell : boolean;
         timeout : array [1..4] of boolean
      end(* secondtrackseqrec *);


var   track : array [trackrange] of trackrec;
      storetodest : array [pathrange] of boolean;
      store : array [storerange] of storerec;
      switch : array [switchrange] of switchrec;
      storetrackseq : array [1..3] of firsttrackseqrec;
      desttrackseq : array [1..4] of secondtrackseqrec;
      tracksempty : array [emptyrange] of boolean;
      puls1hz : boolean;  (* standard timebase *)
      init,auto,restore : boolean;
      i : integer;

procedure timer(inexpr,holdexpr : boolean;
                var timebase,outvar : boolean;
                time : integer); external;

procedure origin(var boolvar : boolean;
                 octaddr : integer); external;

function edge(expr : boolean;
              var edgevar : boolean):boolean; external;

function octal(dec : integer):integer;
begin
   if dec=0 then octal:=0
   else octal:= dec mod 8 + 10*octal(dec div 8);
end; (* octal *)

procedure startof( var dts : secondtrackseqrec);
begin
   with dts do
   begin
      writeln(prr,'start of sect.');
      writeln(prr,'before dest:',(ord(name)+3):3);
      if not init or error
         and (not auto or auto and restore)
         or track[tracknr[2]].engine and timeout[1] and
         track[tracknr[1]].engine and timeout[4]
      then wait:=true;
      if track[tracknr[2]].engine then wait:=false;
      if wait and (storetodest[path[1]] or
         storetodest[path[2]] or storetodest[path[3]] or
         storetodest[path[4]] or storetodest[path[5]]) or
         destfull and timeout[3]
      then track[tracknr[2]].engine:=true;
      if destfull or error or wait
      then track[tracknr[2]].engine:=false;
      track[tracknr[1]].engine:=track[tracknr[2]].engine
         and track[tracknr[2]].verify;
```

```
      timer(track[tracknr[2]].engine,false,
            pulsihz,timeout[1],5);
      timer(track[tracknr[2]].engine and not photocell and
            not storetodest[path[1]] and
            not storetodest[path[2]] and
            not storetodest[path[3]] and
            not storetodest[path[4]] and
            not storetodest[path[5]] and not error,
            false,pulsihz,timeout[4],60);
      timer(track[tracknr[2]].engine and photocell,
            false,pulsihz,timeout[2],5);
      if track[tracknr[2]].engine then destfull:=false;
      if track[tracknr[2]].engine and timeout[1] and
         track[tracknr[1]].verify and not timeout[4]
         and timeout[2]
      then destfull:=true;
      timer(destfull and not photocell,false,
            pulsihz,timeout[3],30);
      if track[tracknr[2]].engine and timeout[1]
         and not track[tracknr[1]].verify
      then error:=true;
      if wait then error:=false;
   end;
end(* startof *);


procedure setswitch(var s : switchrec);

var   i : integer;

begin
   with s do
   begin
      writeln(prr,'set switch no:',(ord(name)+1):3);
      if not init or tracksempty[emptysign]
         and timeout and auto and (level[1]
         and piston[short].pulled and piston[long].pulled
         or level[2]
         and piston[short].pushed and piston[long].pulled
         or level[3]
         and piston[short].pulled and piston[long].pushed
         or level[4]
         and piston[short].pushed and piston[long].pushed)
         or error and (not auto or auto and restore)
      then wait:=true;
      if level[1] or level[2] or level[3] or level[4]
      then wait:=false;
      if wait and auto then
      begin
         if name=sw1 then
         begin
            for i:=1 to switchlevels do
            if storetodest[path[i]] then level[i]:=true;
         end
         else
            for i:=1 to switchlevels do
            if storetodest[path[i]] or
               storetodest[path[4+i]]
            then level[i]:=true;
      end;
      if error or wait then
          for i:=1 to switchlevels do level[i]:=false;
        timer(level[1] or level[2] or level[3] or level[4],
              false,pulsihz,timeout,10);
        if timeout and
           (level[1] and (not piston[short].pulled or
                          not piston[long].pulled) or
            level[2] and (not piston[short].pushed or
                          not piston[long].pulled) or
            level[3] and (not piston[short].pulled or
                          not piston[long].pushed) or
            level[4] and (not piston[short].pushed or
                          not piston[long].pushed))
        then error:=true;
        if wait then error:=false;
        piston[short].pull:=level[1] or level[3];
        piston[short].push:=level[2] or level[4];
        piston[long].pull:=level[1] or level[2];
        piston[long].push:=level[3] or level[4];
   end;
end(* setswitch *);


procedure starting(var sts : firsttrackseqrec);

var   helpvar1,helpvar2 : boolean;

begin
   with sts do
   begin
      writeln(prr,'Starting sect.');
      writeln(prr,'in front of');
      write(prr,'store: ');
      case name of
         st2and4: writeln(prr,'2 and 4');
         st3and5: writeln(prr,'3 and 5');
         st6:     writeln(prr,'6');
      end;
      helpvar1:= not init or track[tracknr[4]].engine and
         not switch[switchnr].error and timeout
         and track[tracknr[2]].verify and
        (track[16].verify and switch[switchnr].level[4] or
         track[15].verify and switch[switchnr].level[3] or
         track[14].verify and switch[switchnr].level[2] or
         track[13].verify and switch[switchnr].level[1] )
         and tracksempty[emptysign] or
         error and (not auto or auto and restore);
      if name=st6 then
      begin
         if helpvar1 then wait:=true;
      end
      else   (* name= st2and4, name= st3and5 *)
         if helpvar1
            and (not track[tracknr[1]].engine or
            track[tracknr[1]].engine and
            track[tracknr[1]].verify )
         then wait:=true;
      if track[tracknr[4]].engine then wait:=false;

      if name=st6 then
      begin
         if wait and auto and
```

```
      (storetodest[path[1]] and track[13].verify
       and switch[switchnr].piston[short].pulled
       and switch[switchnr].piston[long].pulled or
       storetodest[path[2]] and track[14].verify
       and switch[switchnr].piston[short].pushed
       and switch[switchnr].piston[long].pulled or
       storetodest[path[3]] and track[15].verify
       and switch[switchnr].piston[short].pulled
       and switch[switchnr].piston[long].pushed or
       storetodest[path[4]] and track[16].verify
       and switch[switchnr].piston[short].pushed
       and switch[switchnr].piston[long].pushed )
    then track[tracknr[4]].engine:=true;
end
else   (* name= st2and4,name= st3and5 *)
    if wait and auto and
      ((storetodest[path[1]] or storetodest[path[5]])
       and track[13].verify
       and switch[switchnr].piston[short].pulled
       and switch[switchnr].piston[long].pulled or
       (storetodest[path[2]] or storetodest[path[6]])
       and track[14].verify
       and switch[switchnr].piston[short].pushed
       and switch[switchnr].piston[long].pulled or
       (storetodest[path[3]] or storetodest[path[7]])
       and track[15].verify
       and switch[switchnr].piston[short].pulled
       and switch[switchnr].piston[long].pushed or
       (storetodest[path[4]] or storetodest[path[8]])
       and track[16].verify
       and switch[switchnr].piston[short].pushed
       and switch[switchnr].piston[long].pushed )
    then track[tracknr[4]].engine:=true;
if wait or error
then track[tracknr[4]].engine:=false;

if (name=st2and4) or (name=st3and5) then
begin
    if (storetodest[path[1]]
       or storetodest[path[2]]
       or storetodest[path[3]]
       or storetodest[path[4]])
       and track[tracknr[2]].verify
       and track[tracknr[4]].engine
    then track[tracknr[1]].engine:=true;
    if not track[tracknr[2]].verify
    then track[tracknr[1]].engine:=false;
end;

timer(track[tracknr[4]].engine,false,
      pulsihz,timeout,5);

helpvar2:= switch[switchnr].error
    or switch[switchnr].level[1] and
    not track[13].verify
    or switch[switchnr].level[2] and
    not track[14].verify
    or switch[switchnr].level[3] and
    not track[15].verify
    or switch[switchnr].level[4] and
    not track[16].verify;

        if name=st6 then
        begin
           if track[tracknr[4]].engine and timeout and
             (not track[tracknr[4]].verify
              or helpvar2 )
           then error:=true;
        end
        else   (* name= st2and4,name= st3and5 *)
           if track[tracknr[4]].engine and timeout and
             (not track[tracknr[4]].verify
              or track[tracknr[1]].engine and
              not track[tracknr[1]].verify
              or helpvar2 )
           then error:=true;
        if wait then error:=false;

        track[tracknr[3]].engine:=track[tracknr[4]].engine
           and track[tracknr[4]].verify;
        track[tracknr[2]].engine:=track[tracknr[4]].engine
           and track[tracknr[3]].verify;
     end;
end; (* starting *)

procedure openadoorin(var st : storerec);

var   i : integer;

begin
   with st do
   begin
      writeln(prr,'Open doors in');
      writeln(prr,'store:',(ord(name)+2):3);
      if timeout1 and open then close:=true;
      if wait or error then close:=false;
      timer(close,false,pulsihz,timeout2,3);
      if wait and auto
         and track[trackinfrontofstore].verify and
        (storetodest[path[1]] or storetodest[path[2]] or
         storetodest[path[3]] or storetodest[path[4]])
      then open:=true;
      if close then open:=false;
      timer(open,false,pulsihz,timeout1,3);
      if close and timeout2 or not init or
         error and (not auto or auto and restore)
      then wait:=true;
      if open then wait:=false;
      if edge(close,emem) then
      begin
         cerr[4]:=cerr[1] and cerr[2] and cerr[3];
         cerr[3]:=cerr[1] and cerr[2];
         cerr[2]:=cerr[1];
         cerr[1]:=true;
      end;
      if restore or counter
      then for i:=1 to 4 do cerr[i]:=false;
      if cerr[4] then error:=true;
      if wait then error:=false;
      if name=str6 then
         for i:=1 to 6 do
         with door[i] do
         begin
```

```
            if signal and open then opendoor:=true;
            if close then opendoor:=false;
         end
      else
         for i:=1 to nrofdoors do
         with door[i] do
         begin
            if signal and open then opendoor:=true;
            if close then opendoor:=false;
         end;
   end;
end; (* openadoorin *)

procedure initiate;

var   i,j,k : integer;

begin
   (* origin declarations *)

   origin(puls1hz,1001);
   for i:=2 to 20 do
   with track[i] do
   begin
      origin(engine,octal(i-2));
      origin(verify,octal(256+(i-2)));
   end;
   for i:=2 to 6 do
   begin
      if i<>6 then k:=nrofdoors else k:=6;
      for j:=1 to k do
      with store[i].door[j] do
      begin
         origin(opendoor,octal(32+(i-2)*15+(j-1)));
         origin(signal,octal(288+(i-2)*15+(j-1)));
      end;
   end;
   for i:=1 to 3 do
   with switch[i] do
   begin
      with piston[short] do
      begin
         origin(push,octal(112+(i-1)*4+1));
         origin(pull,octal(112+(i-1)*4+2));
         origin(pushed,octal(368+(i-1)*4+1));
         origin(pulled,octal(368+(i-1)*4+2));
      end;
      with piston[long] do
      begin
         origin(push,octal(112+(i-1)*4+3));
         origin(pull,octal(112+(i-1)*4+4));
         origin(pushed,octal(368+(i-1)*4+3));
         origin(pulled,octal(368+(i-1)*4+4));
      end;
   end;
   for i:=1 to 20 do
      origin(storetodest[i],octal(432+(i-1)));
   for i:=1 to 3 do
      origin(tracksempty[i],octal(456+(i-1)));

   (* end of origin *)

   for i:=2 to 6 do
   with store[i] do
   begin
      name:=str2;
      while ord(name)<>(i-2) do name:=succ(name);
      trackinfrontofstore:=i;
      for j:=1 to 4 do path[j]:=(i-2)*4+j;
   end;

   for i:=1 to 3 do
   with storetrackseq[i] do
   begin
      name:=st2and4;
      while ord(name)<>(i-1) do name:=succ(name);
      emptysign:=i;
      switchnr:=(i mod 3)+1;
      if i<>3 then
      begin
         for j:=1 to 8 do path[j]:=((i-1)+(j div 5))*4+j;
         tracknr[1]:=2+(i-1);
         tracknr[2]:=4+(i-1);
         tracknr[3]:=9+(i-1)*2;
         tracknr[4]:=10+(i-1)*2;
      end
      else
      begin
         for j:=1 to 4 do path[j]:=16+j;
         for j:=2 to 4 do tracknr[j]:=6+(j-2);
      end;
   end;

   for i:=1 to 3 do
   with switch[i] do
   begin
      name:=sw1;
      while ord(name)<>(i-1) do name:=succ(name);
      if i=1 then
      begin
         emptysign:=3;
         path:=storetrackseq[3].path;
      end
      else
      begin
         emptysign:=i-1;
         path:=storetrackseq[i-1].path;
      end;
   end;

   for i:=1 to 4 do
   with desttrackseq[i] do
   begin
      name:=dest3;
      while ord(name)<>(i-1) do name:=succ(name);
      for j:=1 to 5 do path[j]:=(j-1)*4+5-i;
      tracknr[1]:=13+(i-1);
      tracknr[2]:=17+(i-1);
   end;
end; (* initiate *)
```

```
begin
   initiate;
   for i:=2 to 6 do openadoorin(store[i]);
   for i:=1 to 3 do starting(storetrackseq[i]);
   for i:=1 to 3 do setswitch(switch[i]);
   for i:=1 to 4 do startof(desttrackseq[i]);
   init:=true;  (* start-up variable in the PBS-program *)
end.
```

```
*C* OPEN DOORS IN
*C* STORE:  2
 AD           1017
 AD           1020
*SSR          1021
 AD           1022
 OR           1023
*RSR          1021
 ADN          1021
 OR           1001
 EDGE         1024
 CNT+         1025
 CNT+         1026
*COT+         1027
 AD            660
 OR            661
 OR            662
 OR            663
*SE           1031
 AD           1031
 AD           1022
 AD           1030
 AD            400
*SSR          1020
 AD           1021
*RSR          1020
 ADN          1020
 OR           1001
 EDGE         1032
 CNT+         1033
 CNT+         1034
*COT+         1017
 AD           1030
 AD           1036
 ORN          1030
*SE           1037
 AD           1037
 AD           1023
 OR           1021
 AD           1027
 ORN          1035
*SSR          1022
 AD           1020
*RSR          1022
 AD           1021
 EDGE         1040
*SSN
 AD           1041
 AD           1042
 AD           1043
*SE           1044
 AD           1041
 AD           1042
*SE           1043
 AD           1041
*SE           1042
*SE           1041
*RS
 AD           1036
 OR           1045
*SSN
*SEN          1041
*SEN          1042
*SEN          1043
*SEN          1044
*RS
 AD           1044
*SSR          1023
 AD           1022
*RSR          1023
 AD            440
 AD           1020
*SSR            40
 AD           1021
*RSR            40
 AD            441
 AD           1020
*SSR            41
 AD           1021
*RSR            41
 AD            442
 AD           1020
*SSR            42
 AD           1021
*RSR            42
 AD            443
 AD           1020
*SSR            43
 AD           1021
*RSR            43
 AD            444
 AD           1020
*SSR            44
 AD           1021
*RSR            44
 AD            445
 AD           1020
*SSR            45
 AD           1021
*RSR            45
 AD            446
 AD           1020
*SSR            46
 AD           1021
*RSR            46
 AD            447
 AD           1020
*SSR            47
 AD           1021
*RSR            47
 AD            450
 AD           1020
*SSR            50
 AD           1021
*RSR            50
 AD            451
 AD           1020
*SSR            51
 AD           1021
*RSR            51
 AD            452
 AD           1020
*SSR            52
 AD           1021
*RSR            52
 AD            453
 AD           1020
*SSR            53
 AD           1021
*RSR            53
 AD            454
 AD           1020
*SSR            54
 AD           1021
*RSR            54
 AD            455
 AD           1020
*SSR            55
 AD           1021
*RSR            55
 AD            456
 AD           1020
*SSR            56
 AD           1021
*RSR            56
*C* OPEN DOORS IN
*C* STORE:  3
 AD           1046
 AD           1047
*SSR          1050
 AD           1051
 OR           1052
*RSR          1050
 ADN          1050
 OR           1001
 EDGE         1053
 CNT+         1054
 CNT+         1055
*COT+         1056
 AD            664
 OR            665
 OR            666
 OR            667
*SE           1057
 AD           1057
 AD           1051
 AD           1030
 AD            401
*SSR          1047
 AD           1050
*RSR          1047
 ADN          1047
 OR           1001
 EDGE         1060
 CNT+         1061
 CNT+         1062
*COT+         1046
 AD           1030
 AD           1036
 ORN          1030
*SE           1063
 AD           1063
 AD           1052
 OR           1050
 AD           1056
 ORN          1035
*SSR          1051
 AD           1047
*RSR          1051
 AD           1050
 EDGE         1064
*SSN
 AD           1065
 AD           1066
 AD           1067
*SE           1070
 AD           1065
 AD           1066
*SE           1067
 AD           1065
*SE           1066
*SE           1065
*RS
 AD           1036
 OR           1071
*SSN
*SEN          1065
*SEN          1066
*SEN          1067
*SEN          1070
*RS
 AD           1070
*SSR          1052
 AD           1051
*RSR          1052
 AD            457
 AD           1047
*SSR            57
 AD           1050
*RSR            57
 AD            460
 AD           1047
*SSR            60
 AD           1050
*RSR            60
 AD            461
 AD           1047
*SSR            61
 AD           1050
*RSR            61
 AD            462
 AD           1047
*SSR            62
 AD           1050
*RSR            62
 AD            463
 AD           1047
*SSR            63
 AD           1050
*RSR            63
 AD            464
 AD           1047
*SSR            64
 AD           1050
```

```
*RSR        64
 AD        465
 AD       1047
*SSR        65
 AD       1050
*RSR        65
 AD        466
 AD       1047
*SSR        66
 AD       1050
*RSR        66
 AD        467
 AD       1047
*SSR        67
 AD       1050
*RSR        67
 AD        470
 AD       1047
*SSR        70
 AD       1050
*RSR        70
 AD        471
 AD       1047
*SSR        71
 AD       1050
*RSR        71
 AD        472
 AD       1047
*SSR        72
 AD       1050
*RSR        72
 AD        473
 AD       1047
*SSR        73
 AD       1050
*RSR        73
 AD        474
 AD       1047
*SSR        74
 AD       1050
*RSR        74
 AD        475
 AD       1047
*SSR        75
 AD       1050
*RSR        75
*C* OPEN DOORS IN
*C* STORE:  4
 AD       1072
 AD       1073
*SSR      1074
 AD       1075
 OR       1076
*RSR      1074
 ADN      1074
 OR       1001
 EDGE     1077
 CNT+     1100
 CNT+     1101
*COT+     1102
 AD        670
 OR        671
 OR        672
 OR        673
*SE       1103
 AD       1103
 AD       1075
 AD       1030
 AD        402
*SSR      1073
 AD       1074
*RSR      1073
 ADN      1073
 OR       1001
 EDGE     1104
 CNT+     1105
 CNT+     1106
*COT+     1072
 AD       1030
 AD       1036
 ORN      1030
*SE       1107
 AD       1107
 AD       1076
 OR       1074
 AD       1102
 ORN      1035
*SSR      1075
 AD       1073
*RSR      1075
 AD       1074
 EDGE     1110
*SSN
 AD       1111
 AD       1112
 AD       1113
*SE       1114
 AD       1111
 AD       1112
*SE       1113
 AD       1111
*SE       1112
*SE       1111
*RS
 AD       1036
 OR       1115
*SSN
*SEN      1111
*SEN      1112
*SEN      1113
*SEN      1114
*RS
 AD       1114
*SSR      1076
 AD       1075
*RSR      1076
 AD        476
 AD       1073
*SSR        76
 AD       1074
*RSR        76
 AD        477
 AD       1073
*SSR        77
 AD       1074
*RSR        77
 AD        500
 AD       1073
*SSR       100
 AD       1074
*RSR       100
 AD        501
 AD       1073
*SSR       101
 AD       1074
*RSR       101
 AD        502
 AD       1073
*SSR       102
 AD       1074
*RSR       102
 AD        503
 AD       1073
*SSR       103
 AD       1074
*RSR       103
 AD        504
 AD       1073
*SSR       104
 AD       1074
*RSR       104
 AD        505
 AD       1073
*SSR       105
 AD       1074
*RSR       105
 AD        506
 AD       1073
*SSR       106
 AD       1074
*RSR       106
 AD        507
 AD       1073
*SSR       107
 AD       1074
*RSR       107
 AD        510
 AD       1073
*SSR       110
 AD       1074
*RSR       110
 AD        511
 AD       1073
*SSR       111
 AD       1074
*RSR       111
 AD        512
 AD       1073
*SSR       112
 AD       1074
*RSR       112
 AD        513
 AD       1073
*SSR       113
 AD       1074
*RSR       113
 AD        514
 AD       1073
*SSR       114
 AD       1074
*RSR       114
*C* OPEN DOORS IN
*C* STORE:  5
 AD       1116
 AD       1117
*SSR      1120
 AD       1121
 OR       1122
*RSR      1120
 ADN      1120
 OR       1001
 EDGE     1123
 CNT+     1124
 CNT+     1125
*COT+     1126
 AD        674
 OR        675
 OR        676
 OR        677
*SE       1127
 AD       1127
 AD       1121
 AD       1030
 AD        403
*SSR      1117
 AD       1120
*RSR      1117
 ADN      1117
 OR       1001
 EDGE     1130
 CNT+     1131
 CNT+     1132
*COT+     1116
 AD       1030
 AD       1036
 ORN      1030
*SE       1133
 AD       1133
 AD       1122
 OR       1120
 AD       1126
 ORN      1035
*SSR      1121
 AD       1117
*RSR      1121
 AD       1120
 EDGE     1134
*SSN
 AD       1135
 AD       1136
 AD       1137
*SE       1140
 AD       1135
 AD       1136
```

```
*SE        1137
 AD        1135
*SE        1136
*SE        1135
*RS
 AD        1036
 OR        1141
*SSN
*SEN       1135
*SEN       1136
*SEN       1137
*SEN       1140
*RS
 AD        1140
*SSR       1122
 AD        1121
*RSR       1122
 AD         515
 AD        1117
*SSR        115
 AD        1120
*RSR        115
 AD         516
 AD        1117
*SSR        116
 AD        1120
*RSR        116
 AD         517
 AD        1117
*SSR        117
 AD        1120
*RSR        117
 AD         520
 AD        1117
*SSR        120
 AD        1120
*RSR        120
 AD         521
 AD        1117
*SSR        121
 AD        1120
*RSR        121
 AD         522
 AD        1117
*SSR        122
 AD        1120
*RSR        122
 AD         523
 AD        1117
*SSR        123
 AD        1120
*RSR        123
 AD         524
 AD        1117
*SSR        124
 AD        1120
*RSR        124
 AD         525
 AD        1117
*SSR        125
 AD        1120
```

```
*RSR        125
 AD         526
 AD        1117
*SSR        126
 AD        1120
*RSR        126
 AD         527
 AD        1117
*SSR        127
 AD        1120
*RSR        127
 AD         530
 AD        1117
*SSR        130
 AD        1120
*RSR        130
 AD         531
 AD        1117
*SSR        131
 AD        1120
*RSR        131
 AD         532
 AD        1117
*SSR        132
 AD        1120
*RSR        132
 AD         533
 AD        1117
*SSR        133
 AD        1120
*RSR        133
*C* OPEN DOORS IN
*C* STORE:  6
 AD        1142
 AD        1143
*SSR       1144
 AD        1145
 OR        1146
*RSR       1144
 ADN       1144
 OR        1001
 EDGE      1147
 CNT+      1150
 CNT+      1151
*COT+      1152
 AD         700
 OR         701
 OR         702
 OR         703
*SE        1153
 AD        1153
 AD        1145
 AD        1030
 AD         404
*SSR       1143
 AD        1144
*RSR       1143
 ADN       1143
 OR        1001
 EDGE      1154
 CNT+      1155
```

```
 CNT+      1156
*COT+      1142
 AD        1030
 AD        1036
 ORN       1030
*SE        1157
 AD        1157
 AD        1146
 OR        1144
 AD        1152
 ORN       1035
*SSR       1145
 AD        1143
*RSR       1145
 AD        1144
 EDGE      1160
*SSN
 AD        1161
 AD        1162
 AD        1163
*SE        1164
 AD        1161
 AD        1162
*SE        1163
 AD        1161
*SE        1162
*SE        1161
*RS
 AD        1036
 OR        1165
*SSN
*SEN       1161
*SEN       1162
*SEN       1163
*SEN       1164
*RS
 AD        1164
*SSR       1146
 AD        1145
*RSR       1146
 AD         534
 AD        1143
*SSR        134
 AD        1144
*RSR        134
 AD         535
 AD        1143
*SSR        135
 AD        1144
*RSR        135
 AD         536
 AD        1143
*SSR        136
 AD        1144
*RSR        136
 AD         537
 AD        1143
*SSR        137
 AD        1144
*RSR        137
 AD         540
```

```
 AD        1143
*SSR        140
 AD        1144
*RSR        140
 AD         541
 AD        1143
*SSR        141
 AD        1144
*RSR        141
*C* STARTING SECT.
*C* IN FRONT OF
*C* STORE: 2 AND 4
 AD        1030
 AD        1036
 ORN       1030
*SE        1176
 AD         416
 AD        1170
 OR         415
 AD        1171
 OR         414
 AD        1172
 OR         413
 AD        1173
*SE        1174
 AD        1174
 AD          10
 ADN       1166
 AD        1167
 AD         402
 AD         710
 ORN       1035
 OR        1176
 AD        1175
*SE        1177
 AD           0
 AD         400
 ORN          0
*SE        1200
 AD        1200
 AD        1177
*SSR       1201
 AD          10
*RSR       1201
 AD         663
 OR         673
*SE        1205
 AD         662
 OR         672
*SE        1204
 AD         661
 OR         671
*SE        1203
 AD         660
 OR         670
*SE        1202
 AD        1202
 AD         413
 AD         566
 AD         570
 OR        1203
```

```
AD        414
AD        565
AD        570
OR       1204
AD        415
AD        566
AD        567
OR       1205
AD        416
AD        565
AD        567
*SE      1206
AD       1206
AD       1201
AD       1030
*SSR       10
AD       1201
OR       1175
*RSR       10
AD        660
OR        661
OR        662
OR        663
*SE      1206
AD       1206
AD        402
AD         10
*SSR        0
ADN       402
*RSR        0
ADN        10
OR       1001
EDGE     1207
CNT+     1210
CNT-     1211
CNT+     1212
*COT+    1167
AD       1173
ADN       413
OR       1166
OR       1172
ADN       414
OR       1171
ADN       415
OR       1170
ADN       416
*SE      1213
AD          0
ADN       400
ORN       410
OR       1213
*SE      1214
AD       1214
AD         10
AD       1167
*SSR     1175
AD       1201
*RSR     1175
AD         10
AD        410
*SE         7
```

```
AD         10
AD        407
*SE         2
*C* STARTING SECT.
*C* IN FRONT OF
*C* STORE: 3 AND 5
AD       1030
AD       1036
ORN      1030
*SE      1225
AD        416
AD       1217
OR        415
AD       1220
OR        414
AD       1221
OR        413
AD       1222
*SE      1223
AD       1223
AD         12
ADN      1215
AD       1216
AD        403
AD        711
ORN      1035
OR       1225
AD       1224
*SE      1226
AD          1
AD        401
ORN         1
*SE      1227
AD       1227
AD       1226
*SSR     1230
AD         12
*RSR     1230
AD        667
OR        677
*SE      1234
AD        666
OR        676
*SE      1233
AD        665
OR        675
*SE      1232
AD        664
OR        674
*SE      1231
AD       1231
AD        413
AD        572
AD        574
OR       1232
AD        414
AD        571
AD        574
OR       1233
AD        415
AD        572
```

```
AD        573
OR       1234
AD        416
AD        571
AD        573
*SE      1235
AD       1235
AD       1230
AD       1030
*SSR       12
AD       1230
OR       1224
*RSR       12
AD        664
OR        665
OR        666
OR        667
*SE      1235
AD       1235
AD        403
AD         12
*SSR        1
ADN       403
*RSR        1
ADN        12
OR       1001
EDGE     1236
CNT+     1237
CNT-     1240
CNT+     1241
*COT+    1216
AD       1222
ADN       413
OR       1215
OR       1221
ADN       414
OR       1220
ADN       415
OR       1217
ADN       416
*SE      1242
AD          1
ADN       401
ORN       412
OR       1242
*SE      1243
AD       1243
AD         12
AD       1216
*SSR     1224
AD       1230
*RSR     1224
AD         12
AD        412
*SE        11
AD         12
AD        411
*SE         3
*C* STARTING SECT.
*C* IN FRONT OF
*C* STORE: 6
```

```
AD       1030
AD       1036
ORN      1030
*SE      1254
AD        416
AD       1246
OR        415
AD       1247
OR        414
AD       1250
OR        413
AD       1251
*SE      1252
AD       1252
AD          6
ADN      1244
AD       1245
AD        404
AD        712
ORN      1035
OR       1254
AD       1253
*SE      1255
AD       1255
*SSR     1256
AD          6
*RSR     1256
AD        700
AD        413
AD        562
AD        564
OR        701
AD        414
AD        561
AD        564
OR        702
AD        415
AD        562
AD        563
OR        703
AD        416
AD        561
AD        563
*SE      1257
AD       1257
AD       1256
AD       1030
*SSR        6
AD       1256
OR       1253
*RSR        6
ADN         6
OR       1001
EDGE     1260
CNT+     1261
CNT-     1262
CNT+     1263
*COT+    1245
AD       1251
ADN       413
OR       1244
```

```
 OR        1250
 ADN        414
 OR        1247
 ADN        415
 OR        1246
 ADN        416
*SE        1264
 ADN        406
 OR        1264
*SE        1265
 AD        1265
 AD           6
 AD        1245
*SSR       1253
 AD        1256
*RSR       1253
 AD           6
 AD         406
*SE           5
 AD           6
 AD         405
*SE           4
*C* SET SWITCH NO:  1
 AD        1030
 AD        1036
 ORN       1030
*SE        1270
 AD        1251
 AD         562
 AD         564
 OR        1250
 AD         561
 AD         564
 OR        1247
 AD         562
 AD         563
 OR        1246
 AD         561
 AD         563
*SE        1267
 AD        1267
 AD         712
 AD        1266
 AD        1030
 ORN       1035
 OR        1270
 AD        1244
*SSR       1271
 AD        1251
 OR        1250
 OR        1247
 OR        1246
*RSR       1271
 AD        1271
 AD        1030
*SE        1004
 AD        1004
*SSN
*RS
 AD         700
*SE        1005
```

```
 AD        1004
 AD        1005
*SSR       1251
 AD        1004
*SSN
*RS
 AD         701
*SE        1005
 AD        1004
 AD        1005
*SSR       1250
 AD        1004
*SSN
*RS
 AD         702
*SE        1005
 AD        1004
 AD        1005
*SSR       1247
 AD        1004
*SSN
*RS
 AD         703
*SE        1005
 AD        1004
 AD        1005
*SSR       1246
 AD        1004
*SSN
*RS
 AD        1244
 OR        1271
*SSN
*SEN       1251
*SEN       1250
*SEN       1247
*SEN       1246
*RS
 AD        1251
 OR        1250
 OR        1247
 OR        1246
*SE        1272
 ADN       1272
 OR        1001
 EDGE      1273
 CNT-      1274
 CNT+      1275
 CNT-      1276
 CNT+      1277
*COT+      1266
 ADN        561
 ORN        563
*SE        1303
 ADN        562
 ORN        563
*SE        1302
 ADN        561
 ORN        564
*SE        1301
 ADN        562
```

```
 ORN        564
*SE        1300
 AD        1300
 AD        1251
 OR        1301
 AD        1250
 OR        1302
 AD        1247
 OR        1303
 AD        1246
*SE        1304
 AD        1304
 AD        1266
*SSR       1244
 AD        1271
*RSR       1244
 AD        1251
 OR        1247
*SE         162
 AD        1250
 OR        1246
*SE         161
 AD        1251
 OR        1250
*SE         164
 AD        1247
 OR        1246
*SE         163
*C* SET SWITCH NO:  2
 AD        1030
 AD        1036
 ORN       1030
*SE        1307
 AD        1173
 AD         566
 AD         570
 OR        1172
 AD         565
 AD         570
 OR        1171
 AD         566
 AD         567
 OR        1170
 AD         565
 AD         567
*SE        1306
 AD        1306
 AD         710
 AD        1305
 AD        1030
 ORN       1035
 OR        1307
 AD        1166
*SSR       1310
 AD        1173
 OR        1172
 OR        1171
 OR        1170
*RSR       1310
 AD        1310
 AD        1030
```

```
*SE        1004
 AD        1004
*SSN
*RS
 AD         660
 OR         670
*SE        1005
 AD        1004
 AD        1005
*SSR       1173
 AD        1004
*SSN
*RS
 AD         661
 OR         671
*SE        1005
 AD        1004
 AD        1005
*SSR       1172
 AD        1004
*SSN
*RS
 AD         662
 OR         672
*SE        1005
 AD        1004
 AD        1005
*SSR       1171
 AD        1004
*SSN
*RS
 AD         663
 OR         673
*SE        1005
 AD        1004
 AD        1005
*SSR       1170
 AD        1004
*SSN
*RS
 AD        1166
 OR        1310
*SSN
*SEN       1173
*SEN       1172
*SEN       1171
*SEN       1170
*RS
 AD        1173
 OR        1172
 OR        1171
 OR        1170
*SE        1311
 ADN       1311
 OR        1001
 EDGE      1312
 CNT-      1313
 CNT+      1314
 CNT-      1315
 CNT+      1316
*COT+      1305
```

```
 ADN      565
 ORN      567
*SE      1322
 ADN      566
 ORN      567
*SE      1321
 ADN      565
 ORN      570
*SE      1320
 ADN      566
 ORN      570
*SE      1317
 AD      1317
 AD      1173
 OR      1320
 AD      1172
 OR      1321
 AD      1171
 OR      1322
 AD      1170
*SE      1323
 AD      1323
 AD      1305
*SSR     1166
 AD      1310
*RSR     1166
 AD      1173
 OR      1171
*SE       166
 AD      1172
 OR      1170
*SE       165
 AD      1173
 OR      1172
*SE       170
 AD      1171
 OR      1170
*SE       167
*C* SET SWITCH NO:  3
 AD      1030
 AD      1036
 ORN     1030
*SE      1326
 AD      1222
 AD       572
 AD       574
 OR      1221
 AD       571
 AD       574
 OR      1220
 AD       572
 AD       573
 OR      1217
 AD       571
 AD       573
*SE      1325
 AD      1325
 AD       711
 AD      1324
 AD      1030
 ORN     1035
```

```
 OR      1326
 AD      1215
*SSR     1327
 AD      1222
 OR      1221
 OR      1220
 OR      1217
*RSR     1327
 AD      1327
 AD      1030
*SE      1004
 AD      1004
*SSN
*RS
 AD       664
 OR       674
*SE      1005
 AD      1004
 AD      1005
*SSR     1222
 AD      1004
*SSN
*RS
 AD       665
 OR       675
*SE      1005
 AD      1004
 AD      1005
*SSR     1221
 AD      1004
*SSN
*RS
 AD       666
 OR       676
*SE      1005
 AD      1004
 AD      1005
*SSR     1220
 AD      1004
*SSN
*RS
 AD       667
 OR       677
*SE      1005
 AD      1004
 AD      1005
*SSR     1217
 AD      1004
*SSN
*RS
 AD      1215
 OR      1327
*SSN
*SEN     1222
*SEN     1221
*SEN     1220
*SEN     1217
*RS
 AD      1222
 OR      1221
 OR      1220
```

```
 OR      1217
*SE      1330
 ADN     1330
 OR      1001
 EDGE    1331
 CNT-    1332
 CNT+    1333
 CNT-    1334
 CNT+    1335
*COT+    1324
 ADN      571
 ORN      573
*SE      1341
 ADN      572
 ORN      573
*SE      1340
 ADN      571
 ORN      574
*SE      1337
 ADN      572
 ORN      574
*SE      1336
 AD      1336
 AD      1222
 OR      1337
 AD      1221
 OR      1340
 AD      1220
 OR      1341
 AD      1217
*SE      1342
 AD      1342
 AD      1324
*SSR     1215
 AD      1327
*RSR     1215
 AD      1222
 OR      1220
*SE       172
 AD      1221
 OR      1217
*SE       171
 AD      1222
 OR      1221
*SE       174
 AD      1220
 OR      1217
*SE       173
*C* START OF SECT.
*C* BEFORE DEST:  3
 AD      1030
 AD      1036
 ORN     1030
*SE      1344
 AD      1344
 AD      1343
 ORN     1035
 OR        17
 AD      1345
 AD        13
 AD      1346
```

```
*SSR     1347
 AD        17
*RSR     1347
 AD       663
 OR       667
 OR       673
 OR       677
 OR       703
*SE      1350
 AD      1350
 AD      1347
 OR      1351
 AD      1352
*SSR       17
 AD      1351
 OR      1343
 OR      1347
*RSR       17
 AD        17
 AD       417
*SE        13
 ADN       17
 OR      1001
 EDGE    1353
 CNT+    1354
 CNT-    1355
 CNT+    1356
*COT+    1345
 AD        17
 ADN     1357
 ADN      663
 ADN      667
 ADN      673
 ADN      677
 ADN      703
 ADN     1343
*SE      1360
 ADN     1360
 OR      1001
 EDGE    1361
 CNT-    1362
 CNT-    1363
 CNT+    1364
 CNT+    1365
 CNT+    1366
 CNT+    1367
*COT+    1346
 AD        17
 AD      1357
*SE      1370
 ADN     1370
 OR      1001
 EDGE    1371
 CNT+    1372
 CNT-    1373
 CNT+    1374
*COT+    1375
 AD        17
*RSR     1351
 AD        17
 AD      1345
```

```
 AD          413
 ADN        1346
 AD         1375
*SSR        1351
 AD         1351
 ADN        1357
*SE         1376
 ADN        1376
 OR         1001
 EDGE       1377
 CNT-       1400
 CNT+       1401
 CNT+       1402
 CNT+       1403
 CNT+       1404
*COT+       1352
 AD           17
 AD         1345
 ADN         413
*SSR        1343
 AD         1347
*RSR        1343
*C* START OF SECT.
*C* BEFORE DEST:  4
 AD         1030
 AD         1036
 ORN        1030
*SE         1406
 AD         1406
 AD         1405
 ORN        1035
 OR           20
 AD         1407
 AD           14
 AD         1410
*SSR        1411
 AD           20
*RSR        1411
 AD          662
 OR          666
 OR          672
 OR          676
 OR          702
*SE         1412
 AD         1412
 AD         1411
 OR         1413
 AD         1414
*SSR          20
 AD         1413
 OR         1405
 OR         1411
*RSR          20
 AD           20
 AD          420
*SE           14
 ADN          20
 OR         1001
 EDGE       1415
 CNT+       1416
 CNT-       1417
```

```
 CNT+       1420
*COT+       1407
 AD           20
 ADN        1421
 ADN         662
 ADN         666
 ADN         672
 ADN         676
 ADN         702
 ADN        1405
*SE         1422
 ADN        1422
 OR         1001
 EDGE       1423
 CNT-       1424
 CNT-       1425
 CNT+       1426
 CNT+       1427
 CNT+       1430
 CNT+       1431
*COT+       1410
 AD           20
 AD         1421
*SE         1432
 ADN        1432
 OR         1001
 EDGE       1433
 CNT+       1434
 CNT-       1435
 CNT+       1436
*COT+       1437
 AD           20
*RSR        1413
 AD           20
 AD         1407
 AD          414
 ADN        1410
 AD         1437
*SSR        1413
 AD         1413
 ADN        1421
*SE         1440
 ADN        1440
 OR         1001
 EDGE       1441
 CNT-       1442
 CNT+       1443
 CNT+       1444
 CNT+       1445
 CNT+       1446
*COT+       1414
 AD           20
 AD         1407
 ADN         414
*SSR        1405
 AD         1411
*RSR        1405
*C* START OF SECT.
*C* BEFORE DEST:  5
 AD         1030
 AD         1036
```

```
 ORN        1030
*SE         1450
 AD         1450
 AD         1447
 ORN        1035
 OR           21
 AD         1451
 AD           15
 AD         1452
*SSR        1453
 AD           21
*RSR        1453
 AD          661
 OR          665
 OR          671
 OR          675
 OR          701
*SE         1454
 AD         1454
 AD         1453
 OR         1455
 AD         1456
*SSR          21
 AD         1455
 OR         1447
 OR         1453
*RSR          21
 AD           21
 AD          421
*SE           15
 ADN          21
 OR         1001
 EDGE       1457
 CNT+       1460
 CNT-       1461
 CNT+       1462
*COT+       1451
 AD           21
 ADN        1463
 ADN         661
 ADN         665
 ADN         671
 ADN         675
 ADN         701
 ADN        1447
*SE         1464
 ADN        1464
 OR         1001
 EDGE       1465
 CNT-       1466
 CNT-       1467
 CNT+       1470
 CNT+       1471
 CNT+       1472
 CNT+       1473
*COT+       1452
 AD           21
 AD         1463
*SE         1474
 ADN        1474
 OR         1001
```

```
 EDGE       1475
 CNT+       1476
 CNT-       1477
 CNT+       1500
*COT+       1501
 AD           21
*RSR        1455
 AD           21
 AD         1451
 AD          415
 ADN        1452
 AD         1501
*SSR        1455
 AD         1455
 ADN        1463
*SE         1502
 ADN        1502
 OR         1001
 EDGE       1503
 CNT-       1504
 CNT+       1505
 CNT+       1506
 CNT+       1507
 CNT+       1510
*COT+       1456
 AD           21
 AD         1451
 ADN         415
*SSR        1447
 AD         1453
*RSR        1447
*C* START OF SECT.
*C* BEFORE DEST:  6
 AD         1030
 AD         1036
 ORN        1030
*SE         1512
 AD         1512
 AD         1511
 ORN        1035
 OR           22
 AD         1513
 AD           16
 AD         1514
*SSR        1515
 AD           22
*RSR        1515
 AD          660
 OR          664
 OR          670
 OR          674
 OR          700
*SE         1516
 AD         1516
 AD         1515
 OR         1517
 AD         1520
*SSR          22
 AD         1517
 OR         1511
 OR         1515
```

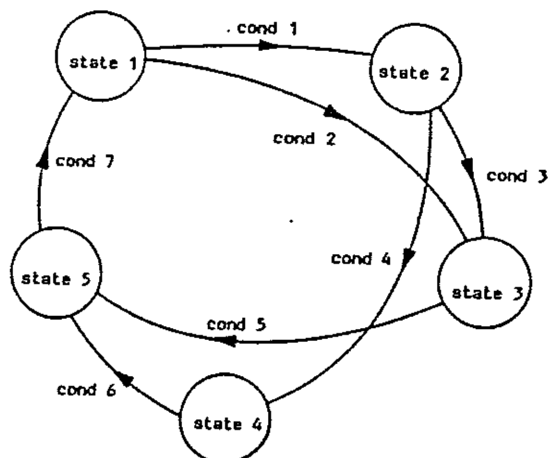
```
*RSR          22
 AD           22
 AD          422
*SE           16
 ADN          22
 OR         1001
 EDGE       1521
 CNT+       1522
 CNT-       1523
 CNT+       1524
*COT+       1513
 AD           22
 ADN        1525
 ADN         660
 ADN         664
 ADN         670
 ADN         674
 ADN         700
 ADN        1511
*SE         1526
 ADN        1526
 OR         1001
 EDGE       1527
 CNT-       1530
 CNT-       1531
 CNT+       1532
 CNT+       1533
 CNT+       1534
 CNT+       1535
*COT+       1514
 AD           22
 AD         1525
*SE         1536
 ADN        1536
 OR         1001
 EDGE       1537
 CNT+       1540
 CNT-       1541
 CNT+       1542
*COT+       1543
 AD           22
*RSR        1517
 AD           22
 AD         1513
 AD          416
 ADN        1514
 AD         1543
*SSR        1517
 AD         1517
 ADN        1525
*SE         1544
 ADN        1544
 OR         1001
 EDGE       1545
 CNT-       1546
 CNT+       1547
 CNT+       1550
 CNT+       1551
 CNT+       1552
*COT+       1520
 AD           22
 AD         1513
 ADN         416
*SSR        1511
 AD         1515
*RSR        1511
*SE         1035
 END
```

7.3 Sekvensstyrning

Sekvensstyrning förekommer ofta i PC-system. Därför är det lämpligt att visa med ett exempel hur det kan lösas i Pascal. Det är lämpligt att använda procedurer för att abstrahera tillståndsövergångarna.

Vårt exempel består av fem tillstånd och sju övergångar enligt nedanstående graf. Tillståndsövergångarna har modellerats med funktionen state. Den fungerar på följande sätt: När systemet når ett nytt tillstånd så ger state värdet true, dvs det går att utföra satserna som hör till det nya tillståndet. Därefter lämnar state värdet false och väntar på att villkoret för någon möjlig tillståndsövergång ska bli sant.

På detta sätt går det att implementera precis så komplicerade sekvensstyrningsprocedurer som behövs i applikationen. Tänkvärt är att spara dem i något slags programbibliotek som alla programmerare använder vid lösningen av sina problem.



```
program Tillstand;

var   st,doing : array [1..5] of boolean;
      cond : array [1..7] of boolean;
      wait : boolean;
      i : integer;

procedure Origin( var boolvar : boolean;
                                address : integer ); external;

function State( var actstate : boolean; condition : boolean;
                var nextstate,waiting : boolean ):boolean;
begin
   State:= false;
   if actstate then
   begin
      if not waiting then
      begin
         State:= true;
         waiting:= true;
      end
      else if condition then
           begin
              actstate:= false;
              nextstate:= true;
              waiting:= false;
           end;
   end;
end (* State *);


begin (* main program *)
   for i:=1 to 5 do Origin(st[i],i-1);
   for i:=1 to 5 do Origin(doing[i],i+9);
   for i:=1 to 7 do Origin(cond[i],i+19);
   Origin(wait,30);
   if State(st[1],cond[1],st[2],wait) or
      State(st[1],cond[2],st[3],wait) then
   begin
      doing[1]:=true;
      (* Other statements which are executed
         when reaching state st[1]             *)
   end;
   if State(st[2],cond[3],st[3],wait) or
      State(st[2],cond[4],st[4],wait) then
   begin
      doing[2]:=true;
      (* Other statements which are executed
         when reaching state st[2]             *)
   end;
   if State(st[3],cond[5],st[5],wait) then
   begin
      doing[3]:=true;
      (* Other statements which are executed
         when reaching state st[3]             *)
   end;
   if State(st[4],cond[6],st[5],wait) then
   begin
      doing[4]:=true;
      (* Other statements which are executed
         when reaching state st[4]             *)
   end;
   if State(st[5],cond[7],st[1],wait) then
   begin
      doing[5]:=true;
      (* Other statements which are executed
         when reaching state st[5]             *)
   end;
end.
```

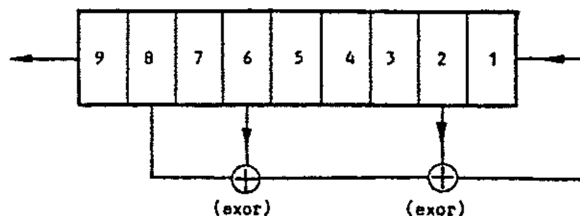
```
 AD       20
*SE     1017
*SEN    1020
 AD        0
*SE     1004
 ADN      30
*SE     1005
 AD     1004
 AD     1005
*SSN
*SE     1020
*SE       30
*RS
 AD     1017
*SE     1006
 AD     1004
 ADN    1005
 AD     1006
*SSN
*SEN       0
*SE        1
*SEN      30
*RS
 AD       21
*SE     1021
*SEN    1022
 AD        0
*SE     1004
 ADN      30
*SE     1005
 AD     1004
 AD     1005
*SSN
*SE     1022
*SE       30
*RS
 AD     1021
*SE     1006
 AD     1004
 ADN    1005
 AD     1006
*SSN
*SEN       0
*SE        2
*SEN      30
*RS
 AD     1020
 OR     1022
*SSR      10
 AD       22
*SE     1022
*SEN    1023
 AD        1
*SE     1004
 ADN      30
*SE     1005
 AD     1004
 AD     1005
*SSN
*SE     1023
*SE       30
*RS
 AD     1022
*SE     1006
 AD     1004
 ADN    1005
 AD     1006
*SSN
*SEN       1
*SE        2
*SEN      30
*RS
 AD       23
*SE     1024
*SEN    1025
 AD        1
*SE     1004
 ADN      30
*SE     1005
 AD     1004
 AD     1005
*SSN
*SE     1025
*SE       30
*RS
 AD     1024
*SE     1006
 AD     1004
 ADN    1005
 AD     1006
*SSN
*SEN       1
*SE        3
*SEN      30
*RS
 AD     1023
 OR     1025
*SSR      11
 AD       24
*SE     1025
*SEN    1026
 AD        2
*SE     1004
 ADN      30
*SE     1005
 AD     1004
 AD     1005
*SSN
*SE     1026
*SE       30
*RS
 AD     1025
*SE     1006
 AD     1004
 ADN    1005
 AD     1006
*SSN
*SEN       2
*SE        4
*SEN      30
*RS
 AD     1026
*SSR      12
 AD       25
*SE     1026
*SEN    1027
 AD        3
*SE     1004
 ADN      30
*SE     1005
 AD     1004
 AD     1005
*SSN
*SE     1027
*SE       30
*RS
 AD     1026
*SE     1006
 AD     1004
 ADN    1005
 AD     1006
*SSN
*SEN       3
*SE        4
*SEN      30
*RS
 AD     1027
*SSR      13
 AD       26
*SE     1027
*SEN    1030
 AD        4
*SE     1004
 ADN      30
*SE     1005
 AD     1004
 AD     1005
*SSN
*SE     1030
*SE      '30
*RS
 AD     1027
*SE     1006
 AD     1004
 ADN    1005
 AD     1006
*SSN
*SEN       4
*SE        0
*SEN      30
*RS
 AD     1030
*SSR      14
 END
```

7.4 Skiftregister

PC-program kan vara lämpliga för modellering av hårdvarufunktioner för t ex undervisning eller uttestning av system. Här har vi programmerat ett lineärt återkopplat skiftregister som kan initieras till ett speciellt tillstånd och sedan skiftas till nästa tillstånd med hjälp av en extern brytare, även initieringen sker med externa brytare. Genom dessa skiftningar kan man studera cykeln hos det modellerade registret. Ändringar göres enkelt via terminal istället för att t ex löda om kretsar m m.

Det går givetvis att modellera hur avancerad hårdvara som helst på det här sättet, bara det går att uttrycka i Pascal och PC-systemets minne räcker till. På detta sätt går det då att testa sina lösningar innan de byggs upp på kretskort.

Nedan visar vi det lineärt återkopplade skiftregister vi modellerat.



```
program Linear;
(* This program is a model of a linear feedback shiftregister *)

const reglength = 9; (* Number of shiftregister cells *)

type  register = array [1..reglength] of boolean;

var   state,switch : register;
      i : 1..reglength;
      nextstate,init,onecycle,e1,e2 : boolean;

function Edge( expr : boolean;
               var cell : boolean ) : boolean; external;

function Exor( expr : boolean;
               var cell : boolean ) : boolean; external;

procedure Origin( var boolvar : boolean;
                      address : integer ); external;

function Octal( d : integer ) : integer;
begin
   if d=0 then Octal:= 0
   else Octal:= d mod 8 + 10*Octal(d div 8);
end (* Octal *);

begin (* Main program *)
   (* Here we define addresses in the pbs-mini corresponding
     to registers state and switch and the switches init and
     onecycle. All the others are allocated in memory by
     pbsgen.
   *)
   for i:=1 to reglength do
   begin
      Origin(switch[i],400+Octal(i-1));
      Origin(state[i],Octal(i));
   end(* for *);
   Origin(init,412); Origin(onecycle,414);

   (* Read in state from switch *)

   if Edge(init,e1)
   then for i:=1 to reglength do state[i]:=switch[i];

   (* Compute next state and display the new state *)

   if Edge(onecycle,e2) then
   begin
      nextstate:=Exor(Exor(state[8],state[6]),state[2]);
      for i:=reglength downto 2 do state[i]:=state[i-1];
      state[1]:=nextstate;
   end;
end.
```

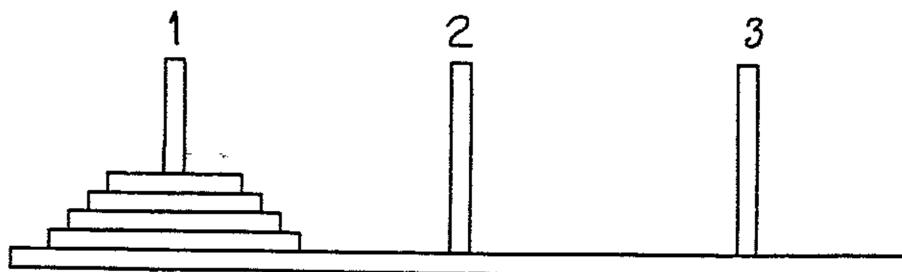
```
 AD        412
 EDGE     1020
*SSN
 AD        400
*SE          1
 AD        401
*SE          2
 AD        402
*SE          3
 AD        403
*SE          4
 AD        404
*SE          5
 AD        405
*SE          6
 AD        406
*SE          7
 AD        407
*SE         10
 AD        410
*SE         11
*RS
 AD        414
 EDGE     1022
*SSN
 AD         10
 EX          6
 EX          2
*SE       1025
 AD         10
*SE         11
 AD          7
*SE         10
 AD          6
*SE          7
 AD          5
*SE          6
 AD          4
*SE          5
 AD          3
*SE          4
 AD          2
*SE          3
 AD          1
*SE          2
 AD       1025
*SE          1
*RS
 END
```

7.5 Towers of Hanoi

Det här exemplet är medtaget mest som kuriosa. Vi vill visa att det går att använda rekursiva procedurer för att beskriva händelserna i en process. Vår kodgenerator tillåter också att det genereras kod från dessa rekursiva procedurer, villkoret för detta är att rekursiviteten måste som alltid vara begränsad och dessutom beräknebar under kodgenereringen.

Processen i det här fallet är egentligen ett spel från Bortre Indien, det kallas för: Towers of Hanoi. Utgångsläget är som bilden visar, alla brickorna på en pinne. Uppgiften är att flytta alla brickorna till pinne nr 3. Endast en bricka får flyttas åt gången och man får endast lägga mindre brickor på större, ej tvärtom. Algoritmen som löser problemet optimalt, dvs med minsta antal flyttningar, är ett typiskt exempel på tillämpningen av rekursivitet. I vårt exempel har vi fyra brickor som ger ett minimalt antal av femton flyttningar.

Maskinen som PBS-mini är tänkt att styra får anses vara utrustad med de faciliteter som omnämnes i Pascal-programmet. Detta för att programmet inte ska bli för plottrigt. I PBS-koden har varje flyttning markerats.



```
program Towerofhanoi(prr);

const discmax = 4;
      nrofmoves = 15;
      (* nrofmoves = 2↑0 + 2↑1 + ... + 2↑(discmax-1) *)

type  pilerec = record
         locatepile,pilelocated : boolean
      end (* pilerec *);

      cranerec = record
         locatetopofpile,topofpilelocated,
         locatedisc,disclocated,
         graspdisc,discgrasped,
         releasedisc,discreleased : boolean
      end (* cranerec *);

      piletype = (source,auxilary,destination);

var   state : array [1..nrofmoves] of boolean;
      frompile,startbutton : boolean;
      control,helpvar,edgehelp : boolean;
      discnr,movecount : integer;
      pile : array [piletype] of pilerec;
      crane : cranerec;

function Edge(expr : boolean;var e : boolean):boolean;external;

procedure Printpile(pilename : piletype);
begin
   case pilename of
   source : write(prr,'source');
   auxilary : write(prr,'aux');
   destination : write(prr,'dest');
   end;
end (* Printpile *);

procedure Moveadisc(from,tohere : piletype);
begin
   writeln(prr,'Move disc nr:',discnr:2,' ');
   Printpile(from);
   write(prr,' -> ');
   Printpile(tohere);
   writeln(prr);
   if state[movecount] then
   begin
      if frompile then
      with crane,pile[from] do
      begin
         locatepile:= not pilelocated;
         locatedisc:= pilelocated and not disclocated;
         graspdisc:= pilelocated and disclocated
                     and not discgrasped;
         frompile:= not discgrasped;
      end
      else
      with crane,pile[tohere] do
      begin
         locatepile:= not pilelocated;
         locatetopofpile:= pilelocated and not
                           topofpilelocated;
         releasedisc:= pilelocated and topofpilelocated
                       and not discreleased;
         if discreleased then
         begin
            if movecount < nrofmoves then
            begin
               state[movecount]:= false;
               movecount:= movecount+1;
               state[movecount]:= true;
               frompile:=true;
            end
            else movecount:= 1;
         end;
      end;
   end;
end (* Moveadisc *);

procedure Move(nrofdiscs : integer;
               from,via,tohere : piletype);
begin
   if nrofdiscs=1 then
   begin
      discnr:=1;
      Moveadisc(from,tohere);
   end
   else
   begin
      Move(nrofdiscs-1,from,tohere,via);
      discnr:=nrofdiscs;
      Moveadisc(from,tohere);
      Move(nrofdiscs-1,via,from,tohere);
   end;
end (* Move *);

begin
   writeln(prr,'Towers of ','Hanoi');
   movecount:=1;
   helpvar:= Edge(startbutton,edgehelp);
   if not control then
   begin
      state[nrofmoves]:= false;
      state[1]:= helpvar;
      control:= helpvar;
      frompile:=true;
   end;
   if control then Move(discmax,source,auxilary,destination);
   if control and state[nrofmoves] then control:= false;
end.
```

```
*C* TOWERS OF HANOI
 AD         1017
 EDGE       1021
*SE         1022
 ADN        1023
*SSN
*SEN        1024
 AD         1022
*SE         1025
 AD         1022
*SE         1023
*SE         1026
*RS
 AD         1023
*SE         1004
 AD         1004
*SSN
*C* MOVE DISC NR: 1
*C* SOURCE -> AUX
*RS
 AD         1025
*SE         1005
 AD         1026
*SE         1006
 AD         1004
 AD         1005
 AD         1006
*SSN
 ADN        1027
*SE         1030
 AD         1027
 ADN        1031
*SE         1032
 AD         1027
 AD         1031
 ADN        1033
*SE         1034
 ADN        1033
*SE         1026
*RS
 AD         1004
 AD         1005
 ADN        1006
*SSN
 ADN        1035
*SE         1036
 AD         1035
 ADN        1037
*SE         1040
 AD         1035
 AD         1037
 ADN        1041
*SE         1042
*RS
 AD         1041
*SE         1007
 AD         1004
 AD         1005
 ADN        1006
 AD         1007
*SSN
*SEN        1025
*SE         1043
*SE         1026
*RS
 AD         1004
*SSN
*C* MOVE DISC NR: 2
*C* SOURCE -> DEST
*RS
 AD         1043
*SE         1005
 AD         1026
*SE         1006
 AD         1004
 AD         1005
 AD         1006
*SSN
 ADN        1027
*SE         1030
 AD         1027
 ADN        1031
*SE         1032
 AD         1027
 AD         1031
 ADN        1033
*SE         1034
 ADN        1033
*SE         1026
*RS
 AD         1004
 AD         1005
 ADN        1006
*SSN
 ADN        1044
*SE         1045
 AD         1044
 ADN        1037
*SE         1040
 AD         1044
 AD         1037
 ADN        1041
*SE         1042
*RS
 AD         1041
*SE         1007
 AD         1004
 AD         1005
 ADN        1006
 AD         1007
*SSN
*SEN        1043
*SE         1046
*SE         1026
*RS
 AD         1004
*SSN
*C* MOVE DISC NR: 1
*C* AUX -> DEST
*RS
 AD         1046
*SE         1005
 AD         1026
*SE         1006
 AD         1004
 AD         1005
 AD         1006
*SSN
 ADN        1035
*SE         1036
 AD         1035
 ADN        1031
*SE         1032
 AD         1035
 AD         1031
 ADN        1033
*SE         1034
 ADN        1033
*SE         1026
*RS
 AD         1004
 AD         1005
 ADN        1006
*SSN
 ADN        1044
*SE         1045
 AD         1044
 ADN        1037
*SE         1040
 AD         1044
 AD         1037
 ADN        1041
*SE         1042
*RS
 AD         1041
*SE         1007
 AD         1004
 AD         1005
 ADN        1006
 AD         1007
*SSN
*SEN        1046
*SE         1047
*SE         1026
*RS
 AD         1004
*SSN
*C* MOVE DISC NR: 3
*C* SOURCE -> AUX
*RS
 AD         1047
*SE         1005
 AD         1026
*SE         1006
 AD         1004
 AD         1005
 AD         1006
*SSN
 ADN        1027
*SE         1030
 AD         1027
 ADN        1031
*SE         1032
 AD         1027
 AD         1031
 ADN        1033
*SE         1034
 ADN        1033
*SE         1026
*RS
 AD         1004
 AD         1005
 ADN        1006
*SSN
 ADN        1035
*SE         1036
 AD         1035
 ADN        1037
*SE         1040
 AD         1035
 AD         1037
 ADN        1041
*SE         1042
*RS
 AD         1041
*SE         1007
 AD         1004
 AD         1005
 ADN        1006
 AD         1007
*SSN
*SEN        1047
*SE         1050
*SE         1026
*RS
 AD         1004
*SSN
*C* MOVE DISC NR: 1
*C* DEST -> SOURCE
*RS
 AD         1050
*SE         1005
 AD         1026
*SE         1006
 AD         1004
 AD         1005
 AD         1006
*SSN
 ADN        1044
*SE         1045
 AD         1044
 ADN        1031
*SE         1032
 AD         1044
 AD         1031
 ADN        1033
*SE         1034
 ADN        1033
*SE         1026
*RS
 AD         1004
 AD         1005
 ADN        1006
*SSN
```

```
 ADN        1027
*SE         1030
 AD         1027
 ADN        1037
*SE         1040
 AD         1027
 AD         1037
 ADN        1041
*SE         1042
*RS
 AD         1041
*SE         1007
 AD         1004
 AD         1005
 ADN        1006
 AD         1007
*SSN
*SEN        1050
*SE         1051
*SE         1026
*RS
 AD         1004
*SSN
*C* MOVE DISC NR: 2
*C* DEST -> AUX
*RS
 AD         1051
*SE         1005
 AD         1026
*SE         1006
 AD         1004
 AD         1005
 AD         1006
*SSN
 ADN        1044
*SE         1045
 AD         1044
 ADN        1031
*SE         1032
 AD         1044
 AD         1031
 ADN        1033
*SE         1034
 ADN        1033
*SE         1026
*RS
 AD         1004
 AD         1005
 ADN        1006
*SSN
 ADN        1035
*SE         1036
 AD         1035
 ADN        1037
*SE         1040
 AD         1035
 AD         1037
 ADN        1041
*SE         1042
*RS
 AD         1041
*SE         1007
 AD         1004
 AD         1005
 ADN        1006
 AD         1007
*SSN
*SEN        1051
*SE         1052
*SE         1026
*RS
 AD         1004
*SSN
*C* MOVE DISC NR: 1
*C* SOURCE -> AUX
*RS
 AD         1052
*SE         1005
 AD         1026
*SE         1006
 AD         1004
 AD         1005
 AD         1006
*SSN
 ADN        1027
*SE         1030
 AD         1027
 ADN        1031
*SE         1032
 AD         1027
 AD         1031
 ADN        1033
*SE         1034
 ADN        1033
*SE         1026
*RS
 AD         1004
 AD         1005
 ADN        1006
*SSN
 ADN        1035
*SE         1036
 AD         1035
 ADN        1037
*SE         1040
 AD         1035
 AD         1037
 ADN        1041
*SE         1042
*RS
 AD         1041
*SE         1007
 AD         1004
 AD         1005
 ADN        1006
 AD         1007
*SSN
*SEN        1052
*SE         1053
*SE         1026
*RS
 AD         1004
*SSN
*C* MOVE DISC NR: 4
*C* SOURCE -> DEST
*RS
 AD         1053
*SE         1005
 AD         1026
*SE         1006
 AD         1004
 AD         1005
 AD         1006
*SSN
 ADN        1027
*SE         1030
 AD         1027
 ADN        1031
*SE         1032
 AD         1027
 AD         1031
 ADN        1033
*SE         1034
 ADN        1033
*SE         1026
*RS
 AD         1004
 AD         1005
 ADN        1006
*SSN
 ADN        1044
*SE         1045
 AD         1044
 ADN        1037
*SE         1040
 AD         1044
 AD         1037
 ADN        1041
*SE         1042
*RS
 AD         1041
*SE         1007
 AD         1004
 AD         1005
 ADN        1006
 AD         1007
*SSN
*SEN        1053
*SE         1054
*SE         1026
*RS
 AD         1004
*SSN
*C* MOVE DISC NR: 1
*C* AUX -> DEST
*RS
 AD         1054
*SE         1005
 AD         1026
*SE         1006
 AD         1004
 AD         1005
 AD         1006
*SSN
 ADN        1035
*SE         1036
 AD         1035
 ADN        1031
*SE         1032
 AD         1035
 AD         1031
 ADN        1033
*SE         1034
 ADN        1033
*SE         1026
*RS
 AD         1004
 AD         1005
 ADN        1006
*SSN
 ADN        1044
*SE         1045
 AD         1044
 ADN        1037
*SE         1040
 AD         1044
 AD         1037
 ADN        1041
*SE         1042
*RS
 AD         1041
*SE         1007
 AD         1004
 AD         1005
 ADN        1006
 AD         1007
*SSN
*SEN        1054
*SE         1055
*SE         1026
*RS
 AD         1004
*SSN
*C* MOVE DISC NR: 2
*C* AUX -> SOURCE
*RS
 AD         1055
*SE         1005
 AD         1026
*SE         1006
 AD         1004
 AD         1005
 AD         1006
*SSN
 ADN        1035
*SE         1036
 AD         1035
 ADN        1031
*SE         1032
 AD         1035
 AD         1031
 ADN        1033
*SE         1034
 ADN        1033
```

```
*SE         1026
*RS
 AD         1004
 AD         1005
 ADN        1006
*SSN
 ADN        1027
*SE         1030
 AD         1027
 ADN        1037
*SE         1040
 AD         1027
 AD         1037
 ADN        1041
*SE         1042
*RS
 AD         1041
*SE         1007
 AD         1004
 AD         1005
 ADN        1006
 AD         1007
*SSN
*SEN        1055
*SE         1056
*SE         1026
*RS
 AD         1004
*SSN
*C* MOVE DISC NR: 1
*C* DEST -) SOURCE
*RS
 AD         1056
*SE         1005
 AD         1026
*SE.        1006
 AD         1004
 AD         1005
 AD         1006
*SSN
 ADN        1044
*SE         1045
 AD         1044
 ADN        1031
*SE         1032
 AD         1044
 AD         1031
 ADN        1033
*SE         1034
 ADN        1033
*SE         1026
*RS
 AD         1004
 AD         1005
 ADN        1006
*SSN
 ADN        1027
*SE         1030
 AD         1027
 ADN        1037
*SE         1040
```

```
 AD         1027
 AD         1037
 ADN        1041
*SE         1042
*RS
 AD         1041
*SE         1007
 AD         1004
 AD         1005
 ADN        1006
 AD         1007
*SSN
*SEN        1056
*SE         1057
*SE         1026
*RS
 AD         1004
*SSN
*C* MOVE DISC NR: 3
*C* AUX -) DEST
*RS
 AD         1057
*SE         1005
 AD         1026
*SE         1006
 AD         1004
 AD         1005
 AD         1006
*SSN
 ADN        1035
*SE         1036
 AD         1035
 ADN        1031
*SE         1032
 AD         1035
 AD         1031
 ADN        1033
*SE         1034
 ADN        1033
*SE         1026
*RS
 AD         1004
 AD         1005
 ADN        1006
*SSN
 ADN        1044
*SE         1045
 AD         1044
 ADN        1037
*SE         1040
 AD         1044
 AD         1037
 ADN        1041
*SE         1042
*RS
 AD         1041
*SE         1007
 AD         1004
 AD         1005
 ADN        1006
 AD         1007
```

```
*SSN
*SEN        1057
*SE         1060
*SE         1026
*RS
 AD         1004
*SSN
*C* MOVE DISC NR: 1
*C* SOURCE -) AUX
*RS
 AD         1060
*SE         1005
 AD         1026
*SE         1006
 AD         1004
 AD         1005
 AD         1006
*SSN
 ADN        1027
*SE         1030
 AD         1027
 ADN        1031
*SE         1032
 AD         1027
 AD         1031
 ADN        1033
*SE         1034
 ADN        1033
*SE         1026
*RS
 AD         1004
 AD         1005
 ADN        1006
*SSN
 ADN        1035
*SE         1036
 AD         1035
 ADN        1037
*SE         1040
 AD         1035
 AD         1037
 ADN        1041
*SE         1042
*RS
 AD         1041
*SE         1007
 AD         1004
 AD         1005
 ADN        1006
 AD         1007
*SSN
*SEN        1060
*SE         1061
*SE         1026
*RS
 AD         1004
*SSN
*C* MOVE DISC NR: 2
*C* SOURCE -) DEST
*RS
 AD         1061
```

```
*SE         1005
 AD         1026
*SE         1006
 AD         1004
 AD         1005
 AD         1006
*SSN
 ADN        1027
*SE         1030
 AD         1027
 ADN        1031
*SE         1032
 AD         1027
 AD         1031
 ADN        1033
*SE         1034
 ADN        1033
*SE         1026
*RS
 AD         1004
 AD         1005
 ADN        1006
*SSN
 ADN        1044
*SE         1045
 AD         1044
 ADN        1037
*SE         1040
 AD         1044
 AD         1037
 ADN        1041
*SE         1042
*RS
 AD         1041
*SE         1007
 AD         1004
 AD         1005
 ADN        1006
 AD         1007
*SSN
*SEN        1061
*SE         1024
*SE         1026
*RS
 AD         1004
*SSN
*C* MOVE DISC NR: 1
*C* AUX -) DEST
*RS
 AD         1024
*SE         1005
 AD         1026
*SE         1006
 AD         1004
 AD         1005
 AD         1006
*SSN
 ADN        1035
*SE         1036
 AD         1035
 ADN        1031
```

```
*SE     1032
 AD     1035
 AD     1031
 ADN    1033
*SE     1034
 ADN    1033
*SE     1026
*RS
 AD     1004
 AD     1005
 ADN    1006
*SSN
 ADN    1044
*SE     1045
 AD     1044
 ADN    1037
*SE     1040
 AD     1044
 AD     1037
 ADN    1041
*SE     1042
*RS
 AD     1041
*SE     1007
 AD     1004
 AD     1005
 ADN    1006
 AD     1007
*SSN
*RS
 AD     1004
*SSN
*RS
 AD     1023
 AD     1024
*RSR    1023
 END
```

REFERENSER MED KOMMENTARER

1. K. Jensen and N. Wirth, Pascal - User Manual and Report, Lecture Notes in Computer Science 18, Springer Verlag, 1974

   Om något kan kallas för "standard" Pascal, när detta skrivs, så definieras det av denna rapport.

2. Pascal: The Language and its Implementation, edited by D.W. Barron, J. Wiley, 1981

   Denna bok är en samling av olika "klassiska" artiklar om Pascal och olika implementeringar. En del av artiklarna finns ej tillgängliga på annan plats än i den här boken.

3. A.J. Hurst, Pascal-P, Program Structure and Program Behaviour, Software - Practice and Experience 10, 1029-1036 (1980)

   Artikeln innehåller en del om P-kompilatorn och dess annorlunda implemention på en mindre dator, DG Nova.

4. P. Kornerup et al., Interpretation and Code Generation Based on Intermediate Languages, Software - Practice and Experience 10, 635-658 (1980)

   Här beskrivs en ganska avancerad form av mellankod som innehåller mycket mer information än den vi nyttjat. En dylik mellankod kan både förenkla och försvåra olika typer av kodgenerering.

5. R.N. Faiman and A.A. Kortesoja, An Optimizing Pascal Compiler, IEEE Trans. Soft. Eng. SE-6, 512-519 (nov 1980)

   Artikeln behandlar hur optimering kan inkluderas i en Pascal-kompilator.

6. R.E. Berry, Experience with the Pascal P-Compiler, Software - Practice and Experience 8, 617-627 (1978)

Behandlar utvecklingen av en bättre och snabbare variant av den ursprungliga P-kompilatorn. Arbetet gjordes i Lancaster, England och är alltså den P-kompilator vi har använt.

7. R.E. Berry et al., Lancaster Nova Pascal, Dep. Comp. Stud., Univ. Lancaster, ref: CA/78/51 (feb 1978)

Beskriver i detalj den P-kompilatorn vi använt. Även interpretator m m behandlas i häftet.

8. Lancaster Nova Pascal User's Guide, Release 2 Update 0, Dep. Comp. Stud., Univ. Lancaster, ref: CSD AF014

Beskriver det Pascal-system som vi använde på DG Eclipse i början av projektet. Det körs under operativsystemet RDOS.

9. K. Ericson och H. Lunell, Redskap för kompilatorframställning, Datalogicentrum, Linköping, LiTH-MAT-R-80-39 (1980)

Rapporten behandlar hela området från parsers till kodgeneratorer. Vårt intresse rörde mest mellankoder och deras utformning, samt kodgenerering.

10. P. Emanuelson och A. Haraldsson, On Compiling Embedded Languages in Lisp, Datalogicentrum, Linköping, LiTH-MAT-R-80-20 (1980)

Här behandlas hur partialevaluering gjorts för någon form av pattern-matcher i Lisp.

11. A. Ström, Experiment med partialevaluering, Datalogicentrum, Linköping, LiTH-MAT-R-79-42 (1979)

Beskriver en del experiment med interaktiva program och partiell evaluering i dem.

12. L. Beckman et al, A Partial Evaluator, and its Use as a Programming Tool, Artificial Intelligence 7, 319-357 (1976)

En mer grundläggande artikel om partialevaluering och dess ursprung, funktion m m.

13. H. Takahashi, An Automatic-Controller Description Language, IEEE Trans. on Softw. Engin. 6, 53-64 (1980)

Författaren till artikeln diskuterar hur ett styrspråk kan utformas. Olika typer av beskrivningar behandlas.

14. D.R. Hanson, Code Improvement via Lazy Evaluation, Inf. Proc. Letters 11, 163-167 (1980)

Artikeln behandlar hur det går att förbättra koden för "expressions", att jämföra med vår hantering av dem i kodgeneratorn.

15. A.V. Aho and J.D. Ullman, Principles of Compiler Design, Addison-Wesley 1977

Grundläggande teoretiskt om hur kompilatorer fungerar och konstrueras.

16. PBS mini Teknisk beskrivning, SATT-Electronics, Malmö, MINI.130.01.Sv (1980)

Beskriver PBS-mini och dess instruktionsrepertoar, samt lite om PBS-mini:s funktion.

17. PC-system En allmän introduktion, SATT-Electronics, Malmö, (1980)

Allmänt om PC-system och deras funktion och användning.

18. PBS DOC 10, SATT-Electronics, Malmö, DOC10.130.01.Sv (1980)

Beskrivning av och manual till det dokumentationssytem för PBS-mini som körs på ABC-80.

19. PS/PROG 11, SATT-Electronics, Malmö, PS/PROG11.130.01.Sv

Manual till PBS-mini:s programmeringsenheter.

APPENDIX 1

Sammanställning av

- Använt subset av Pascal
- Pascal-element som interpreteras
- Pascal-element som ger upphov till PC-kod

Datatyper

Enkla typer:

- Boolean:
  Variabler av typen boolean motsvaras direkt av minnesceller eller in- och ut-gångar i PBS-mini:n. Ett undantag är formella parametrar till procedurer, se nedan. Detta innebär att variabler av denna typ och operationer på dem kommer att översättas till PBS-kod. Kodgeneratorn uppfattar alltid boolska variablers värden som okända vilket innebär att de aldrig evalueras till något värde under kodgenereringen. Boolska värden och operationer på dessa evalueras dock.

- Integer:
  Variabler av typen integer har ingen motsvarighet i PBS-mini, varför dessa kommer att evalueras under kodutläggningen. Även operationer på dem evalueras.

- Real:
  Ej implementerat.

- Char:
  Ej implementerat.

- Pekare:
  Ej implementerat.

- Egendefinierade skalära typer:
  Behandlas på samma sätt som integers, d v s evalueras.

Strukturerade typer:

- Record:
  Används på vanligt sätt i Pascal.

- Array:
  Används på vanligt sätt, dock måste index vara evaluerbart under kodutläggningen.

- Set:
  Ej implementerat.

- File:
  Ej implementerat.

Satser

- Tilldelningssatser:
  Om tilldelningssatsen har en boolsk variabel till vänster om tilldelningstecknet, så kommer kod att genereras. I annat fall interpreteras satsen.

- If-satser:
  Om villkorsuttrycket är evaluerbart under kodgenereringen så kommer kod att läggas ut för den gren uttryckets värde stipulerar. Skulle så icke vara fallet genereras kod motsvarande if-satsen.

- Case-satser:
  Case-satsen interpreteras, och kan således inte appliceras på boolean.

- While-satser:
  Interpreteras, vilket medför att villkors-uttrycket måste vara evaluerbart under kodgenereringen.

- Repeat-satser:
  Se while-satser ovan.

- For-satser:
  For-satsen interpreteras, varför styrande variabelns start respektive slutvärde måste vara evaluerbara.

- Goto-satser:
  Ej implementerat.

Procedurer och funktioner

- Parameter-överföringen:
  All parameteröverföring fungerar som i vanlig Pascal, dock med ett undantag: värdeanropade boolska variabler. Om det skickas uttryck, som inte är evaluerbara under kodgenereringen, eller vanliga variabler som parametrar till en procedur, vars parameterlista innehåller värdeanropade boolska variabler, så kommer det att genereras PBS-kod för värdekopieringen. Om den aktuella parametern är ett uttryck som är evaluerbart under kodgenereringen (t ex en boolsk konstant), så blir det ingen PBS-kod utan det kommer att interpreteras ända tills dess att den formella parametern tilldelas ett värde i procedur-kroppen.

- Egen-definierade procedurer:
  Dessa interpreteras precis som programmet i övrigt. Funktioner lämnar värden precis som i vanlig Pascal, om funktionen är av typen boolean så kommer anropet att ge upphov till PBS-kod precis som för boolean i övrigt.

- Standard och systemprocedurer:
  Alla Pascals standardprocedurer är ej implementerade. Följande procedurer med parameter av typen integer eller egendefinierad skalär typ finns dock: succ(), pred(), ord(). Systemprocedurerna används som tidigare beskrivits för att överföra information till kodgeneratorn.

In- och ut-matning

- Kommentarer i PBS-koden:
  Genom att specificera filnamnet PRR i programhuvudet så kan PBS-koden som genereras förses med kommentarer. Detta görs med write och writeln på vanligt Pascal-maner. Argumenten till write(ln)-satserna kan vara integer-variabler, integer-konstanter och text-strängar omslutna av apostrofer.

- Interaktiv kodgenerering:
  Det finns möjlighet till in- och ut-matning till terminalen man arbetar vid under kodgenereringen. Detta kan fås genom att specificera filnamnen INPUT och OUTPUT i programhuvudet. Då kan man kommunicera med det Pascal-program som det ska genereras PBS-kod för och styra kodutläggningen med lämpliga read och write-satser. Endast integers kan läsas in och för write-satserna gäller som ovan för kommentarerna.

APPENDIX 2

Listning av den implementerade kodgeneratorn

```
program pbsgen(pcodein,output,pbsout,input);


(**********************************************************
*                                                         *
*                                                         *
*     Code generator for SATT Electronics PBS-mini        *
*                                                         *
*     Runable on VAX 11/780, VMS version 2.3              *
*                                                         *
*     Input:   Pascal P-code                              *
*     Output:  PC code for PBS-mini                       *
*                                                         *
*     Authors: Sten Minor                                 *
*              Oskar Permvall                             *
*                                                         *
*                                                         *
*     Date:    June 15, 1981                              *
*                                                         *
*     Department of automatic control                     *
*     Lund Institute of Technology                        *
*                                                         *
**********************************************************)


const slimit = 500;        (* upper limit of compiler stack *)
      numbinstr = 64;      (* number of p-code instructions *)
      lghtinstr = 5;       (* lenght of p-code instruction  *)
      pbsnumbinstr = 19;(* number of pbs instructions       *)
      pbslow = -1;         (* =octal(palow), -1 error value *)
      pbshigh = 1777;      (* =octal(pahigh)                *)
      palow = -1;          (* lower bound for pbs-addresses *)
      pamidlow = 526;      (* pbsaddress nr 1016 decimal    *)
      pahigh = 1023;       (* pbsaddress nr 1777 decimal    *)
      pbsifmin = 1004;     (* lower address of pbs if area  *)
      ifmax = 10;          (* maximum of nested if          *)
      llimit = 128;        (* maximum of labels in pcode    *)
      maxtxt = 16;         (* max length of text element    *)
      cprd = 'PRD          ';
             (* identifies prdfile in open statement        *)
      cprr = 'PRR          ';
             (* identifies prr file in open statement       *)
      cout = 'OUTPUT       ';
             (* identifies output file in open stat         *)


type  titype = (mst,addr,val,expr,noitem,fil);
                                    (* type of stack-item *)
      tlegal = (noend,lastend,both);
      pbsrange = pbslow..pbshigh;
      tfil = (infile,outfile,pbsfile);
      titem = record                (* item in stack *)
         neg : boolean;
         case itype : titype of
            mst :    (return : integer;
                      statlink : integer;
                      dynlink : integer);
            addr :   (address : pbsrange);
            val :    (valu:integer);
            fil :    ( ftype : tfil );
            expr :   (first : ↑tnode;
                      last : ↑tnode;
                      orlegal : tlegal;
                      andlegal : tlegal);
            noitem :();
      end(* record *);

      tltype = (cxp,cup,prog,store,nolab);
                                         (* type of label *)
      tproctype = (boolproc,intproc,procproc);
      tcxp = (timer,origin,edge,exor,pbsadof,clearpbs);
                                 (* system procedure names *)
      tlab = record
         level : integer;
         proctype : tproctype;
         case ltype : tltype of
            cxp : (name : tcxp);
            cup,prog,store,nolab : (dummy:integer);
      end (* tlab *);

      taddr = (free,used,usedbefore);
                   (* indicates if a pbsaddress is used *)
      parange = palow..pahigh;

      tlgth = 1..maxtxt;
      ctext = packed array [tlgth] of char;
      ptext = ↑ctext;
      pch = ↑char;
      ttxtype = (chr,tex);
      ptxtnode = ↑ttxtnode;
      ttxtnode = record
         tnr : integer;
         next : ptxtnode;
         case txtype : ttxtype of
            chr : (c:pch);
            tex : (t:ptext);
      end(* record *);

      pleavenode = ↑tleavenode;
      tleavenode = record
         next : pleavenode;
         val : pbsrange
      end(* record *);

      nodep = ↑tnode;
      tnode = record                (* node in expression list *)
         nodeinstr : integer;
         nodearg   : pbsrange;
         next  : nodep
      end(* record *);

      tstack = array [0..slimit] of titem;

      tinstr = (nosym,ldobsym,ldcbsym,srobsym,andsym,iorsym,
                notsym,ifsym,thensym,elsesym,eifsym,laosym,
                ldcisym,chkisym,stobsym,decisym,incisym,
                ixasym,ldoisym,sroisym,adisym,dvisym,mpisym,
                sbisym,modsym,equisym,grtisym,geqisym,
```

```
            lesisym,leqisym,neqisym,cupsym,retbsym,
            retpsym,indbsym,mstsym,strbsym,lodasym,
            entsym,lodbsym,ldasym,incasym,strasym,
            strisym,lsym,fjpsym,ujpsym,movsym,lodisym,
            pcodesym,retisym,stpsym,indisym,ujcsym,
            xjpsym,ngisym,stoisym,cspsym,txtsym,
            cxpsym,ldoasym,ldccsym,endsym);

      tset = set of tinstr;

      targ = (int,int2,lbl,intlbl,jlbl,proc,none,name,txt);

      tpctext = packed array [1..lghtinstr] of char;

      terror = (esp,eorg,ecase,eif,echki,epcaddr,eitem,
               elab,el,efil,etxt,eproc,eext);

      tpcrec = record         (* item in pcfile and symfile *)
         pcsym : tinstr;
         asym : targ;
         a1,a2 : integer
      end (* record *);


var   itxt : array [1..numbinstr] of tpctext;
      isym : array [1..numbinstr] of tinstr;
      iarg : array [1..numbinstr] of targ;
      pbstxt : array [1..pbsnumbinstr] of
                 packed array [1..4] of char;
      pbsarg : array [1..pbsnumbinstr] of boolean;
      pbsaddr : array [parange] of taddr; (* addresspool *)
      l : array [1..llimit] of tlab;

                     (* variables returned by insymbol: *)
      nextinstsym,instsym : tinstr;(* p-code instruction *)
      nextargsym,argsym : targ;      (* type of arguments *)
      nextarg1,arg1 : integer;(* value of first argument *)
      nextarg2,arg2 : integer;(* value of second argument*)

      pbsout,pcodein : text;
      symfile : file of tpcrec;
                  (* file used between scan1 and scan2 *)
      pcfile : file of tpcrec;
            (* file containing the "cleaned" pcodeinfile *)
      pccount : integer;
               (* pointer into pcfile,
                  pointing to the instruction just read *)
      pctext : tpctext;
            (* used for input of pcode from pcodeinfile *)
      pcmaxaddr : integer;    (* nr of elements in pcfile *)

      s : tstack;                  (* code-generator stack *)
      sp : integer;                (* stackpointer         *)
      mp : integer;                (* markpointer          *)
      aplowbound : parange;
                (* to switch between pamidlow and palow,
                         used in function getaddress *)
      ap,rp : parange;             (* addresspointers *)
      returnaddresses : boolean;
                      (* control for proc. leaveaddress *)
      errflag : boolean;
```

```
              (* becomes true when entering procedure error *)
      wlnflgprr : boolean;
                 (* control for not mixing instructions
                             and comments in file prr *)
      txtpool : record
         first,last : ptxtnode
      end;
      tcount : integer;             (* =txtpool.last↑.tnr *)
      semaphor : integer;
      leavepool : record
         ap,rp,fp : pleavenode
      end;
      iflev : integer;              (* current level of if *)
      ifarr : array [1..ifmax] of (thenstat,elsestat);
      pcaddr : integer;            (* remembering pccount  *)
      pool : record                (* pool of 'expr'-nodes *)
         first,last : nodep
      end;


(* - - initialization - - - - - - - - - - - - - - - - - -*)


procedure initsimple;

var   i : integer;

begin
   sp:=-1;
   mp:=0;
   pccount:=0;
   iflev:=0;
   returnaddresses:=true;
   for i:=palow to pahigh do pbsaddr[i]:=free;
   for i:=512 to pamidlow do pbsaddr[i]:=used;
   rp:=pahigh;
   ap:=pamidlow;
   aplowbound:=pamidlow;
   for i:=1 to llimit do
   begin
      l[i].level:=-1;
      l[i].ltype:=nolab;
      l[i].proctype:=procproc;
   end(* for *);
   for i:=0 to slimit do
   with s[i] do
   begin
      neg:=false;
      itype:=noitem;
   end;
   with pool do
   begin
      first:=nil;
      last:=nil;
   end(* with *);
   semaphor:=-1;
   with leavepool do
   begin
```

```
    new(rp);
    ap:=rp;
    fp:=rp;
    ap^.next:=nil;
  end(* with *);
  txtpool.first:=nil;
  txtpool.last:=nil;
  errflag:=false;
  wlnflgprr:=true;
  tcount:=0;
end (* initsimple *);


procedure initarray;
begin

                  (* p-code instructions used in inline *)
  itxt[1]:= ' LDOB'; isym[1]:= ldobsym; iarg[1]:= int ;
  itxt[2]:= ' LDCB'; isym[2]:= ldcbsym; iarg[2]:= int ;
  itxt[3]:= ' SROB'; isym[3]:= srobsym; iarg[3]:= int ;
  itxt[4]:= ' AND '; isym[4]:= andsym ; iarg[4]:= none;
  itxt[5]:= ' IOR '; isym[5]:= iorsym ; iarg[5]:= none;
  itxt[6]:= ' NOT '; isym[6]:= notsym ; iarg[6]:= none;
  itxt[7]:= ' IF  '; isym[7]:= ifsym  ; iarg[7]:= none;
  itxt[8]:= ' THEN'; isym[8]:= thensym; iarg[8]:= none;
  itxt[9]:= ' ELSE'; isym[9]:= elsesym; iarg[9]:= none;
  itxt[10]:=' EIF '; isym[10]:=eifsym ; iarg[10]:=none;
  itxt[11]:=' LAO '; isym[11]:=laosym ; iarg[11]:=int ;
  itxt[12]:=' LDCI'; isym[12]:=ldcisym; iarg[12]:=int ;
  itxt[13]:=' CHKI'; isym[13]:=chkisym; iarg[13]:=int2;
  itxt[14]:=' STOB'; isym[14]:=stobsym; iarg[14]:=none;
  itxt[15]:=' DECI'; isym[15]:=decisym; iarg[15]:=int ;
  itxt[16]:=' INCI'; isym[16]:=incisym; iarg[16]:=int ;
  itxt[17]:=' IXA '; isym[17]:=ixasym ; iarg[17]:=int ;
  itxt[18]:=' LDOI'; isym[18]:=ldoisym; iarg[18]:=int ;
  itxt[19]:=' SROI'; isym[19]:=sroisym; iarg[19]:=int ;
  itxt[20]:=' ADI '; isym[20]:=adisym ; iarg[20]:=none;
  itxt[21]:=' DVI '; isym[21]:=dvisym ; iarg[21]:=none;
  itxt[22]:=' MPI '; isym[22]:=mpisym ; iarg[22]:=none;
  itxt[23]:=' SBI '; isym[23]:=sbisym ; iarg[23]:=none;
  itxt[24]:=' MOD '; isym[24]:=modsym ; iarg[24]:=none;
  itxt[25]:=' EQUI'; isym[25]:=equisym; iarg[25]:=none;
  itxt[26]:=' GRTI'; isym[26]:=grtisym; iarg[26]:=none;
  itxt[27]:=' GEQI'; isym[27]:=geqisym; iarg[27]:=none;
  itxt[28]:=' LESI'; isym[28]:=lesisym; iarg[28]:=none;
  itxt[29]:=' LEQI'; isym[29]:=leqisym; iarg[29]:=none;
  itxt[30]:=' NEQI'; isym[30]:=neqisym; iarg[30]:=none;
  itxt[31]:=' CUP '; isym[31]:=cupsym ; iarg[31]:=intlbl;
  itxt[32]:=' RETB'; isym[32]:=retbsym; iarg[32]:=none;
  itxt[33]:=' RETP'; isym[33]:=retpsym; iarg[33]:=none;
  itxt[34]:=' INDB'; isym[34]:=indbsym; iarg[34]:=int ;
  itxt[35]:=' MST '; isym[35]:=mstsym ; iarg[35]:=int ;
  itxt[36]:=' STRB'; isym[36]:=strbsym; iarg[36]:=int2;
  itxt[37]:=' LODA'; isym[37]:=lodasym; iarg[37]:=int2;
  itxt[38]:=' ENT '; isym[38]:=entsym ; iarg[38]:=intlbl;
  itxt[39]:=' LODB'; isym[39]:=lodbsym; iarg[39]:=int2;
  itxt[40]:=' LDA '; isym[40]:=ldasym ; iarg[40]:=int2;
  itxt[41]:=' INCA'; isym[41]:=incasym; iarg[41]:=int ;
  itxt[42]:=' STRA'; isym[42]:=strasym; iarg[42]:=int2;
  itxt[43]:=' STRI'; isym[43]:=strisym; iarg[43]:=int2;
  itxt[44]:='L    '; isym[44]:=lsym   ; iarg[44]:=lbl ;
  itxt[45]:=' FJP '; isym[45]:=fjpsym ; iarg[45]:=jlbl;
  itxt[46]:=' UJP '; isym[46]:=ujpsym ; iarg[46]:=jlbl;
  itxt[47]:=' MOV '; isym[47]:=movsym ; iarg[47]:=int ;
  itxt[48]:=' LODI'; isym[48]:=lodisym; iarg[48]:=int2;
  itxt[49]:='PCODE'; isym[49]:=pcodesym;iarg[49]:=none;
  itxt[50]:=' RETI'; isym[50]:=retisym; iarg[50]:=none;
  itxt[51]:=' STP '; isym[51]:=stpsym ; iarg[51]:=none;
  itxt[52]:=' .ENT'; isym[52]:=nosym  ; iarg[52]:=proc;
  itxt[53]:=' UJC '; isym[53]:=ujcsym ; iarg[53]:=none;
  itxt[54]:=' XJP '; isym[54]:=xjpsym ; iarg[54]:=jlbl;
  itxt[55]:=' NGI '; isym[55]:=ngisym ; iarg[55]:=none;
  itxt[56]:=' INDI'; isym[56]:=indisym; iarg[56]:=int ;
  itxt[57]:=' STOI'; isym[57]:=stoisym; iarg[57]:=none;
  itxt[58]:=' CSP '; isym[58]:=cspsym ; iarg[58]:=name;
  itxt[59]:=' .TXT'; isym[59]:=txtsym ; iarg[59]:=txt ;
  itxt[60]:=' CXP '; isym[60]:=cxpsym ; iarg[60]:=intlbl;
  itxt[61]:=' .EXT'; isym[61]:=nosym  ; iarg[61]:=proc;
  itxt[62]:=' LDOA'; isym[62]:=ldoasym; iarg[62]:=int ;
  itxt[63]:=' LDCC'; isym[63]:=ldccsym; iarg[63]:=txt ;
  itxt[numbinstr]:=' .END'; isym[numbinstr]:=endsym;
  iarg[numbinstr]:=none;

end(* initarray *);

procedure init2array;
begin

                 (* pbs instructions used in genpbs *)
  pbstxt[1]:= 'AD  ';  pbsarg[1]:= true ;
  pbstxt[2]:= 'ADN ';  pbsarg[2]:= true ;
  pbstxt[3]:= 'OR  ';  pbsarg[3]:= true ;
  pbstxt[4]:= 'ORN ';  pbsarg[4]:= true ;
  pbstxt[5]:= 'SE  ';  pbsarg[5]:= true ;
  pbstxt[6]:= 'SEN ';  pbsarg[6]:= true ;
  pbstxt[7]:= 'SS  ';  pbsarg[7]:= false;
  pbstxt[8]:= 'SSN ';  pbsarg[8]:= false;
  pbstxt[9]:= 'RS  ';  pbsarg[9]:= false;
  pbstxt[10]:='CNT-';  pbsarg[10]:=true ;
  pbstxt[11]:='CNT+';  pbsarg[11]:=true ;
  pbstxt[12]:='COT-';  pbsarg[12]:=true ;
  pbstxt[13]:='COT+';  pbsarg[13]:=true ;
  pbstxt[14]:='EDGE';  pbsarg[14]:=true ;
  pbstxt[15]:='EX  ';  pbsarg[15]:=true ;
  pbstxt[16]:='AP  ';  pbsarg[16]:=false;
  pbstxt[17]:='RP  ';  pbsarg[17]:=false;
  pbstxt[18]:='CP  ';  pbsarg[18]:=false;
  pbstxt[pbsnumbinstr]:='END ';
  pbsarg[pbsnumbinstr]:=false;
end (* init2array *);


procedure fileinit;

type  namestring = packed array [1..13] of char;
      filetype = (inp,outp);

var   inputname,outputname : namestring;
      status : integer;
      test : boolean;
      ch : char;
```

```
procedure filename(n : packed array [integer] of char;
                       var st : integer); extern;

procedure filecontrol(var name : namestring;
              var test : boolean; ftype : filetype);

var   status,nrofchars,maxnrofchars : integer;
      dotincluded : boolean;

begin
   name:='            ';
   maxnrofchars:=9;
   nrofchars:=1;
   dotincluded:=false;
   while (nrofchars<=maxnrofchars) and not eoln do
   begin
      read(ch);
      name[nrofchars]:=ch;
      if ch='.' then
      begin
         dotincluded:=true;
         maxnrofchars:=13;
      end;
      nrofchars:=nrofchars+1;
   end;
   readln;
   if ftype=inp then
   begin
      if not dotincluded then
      begin
         name[nrofchars]:='.';
         name[nrofchars+1]:='p';
         name[nrofchars+2]:='c';
         name[nrofchars+3]:='d';
      end;
      filename(name,status);
      test:=status>=0;
   end
   else  if not dotincluded then
         begin
            name[nrofchars]:='.';
            name[nrofchars+1]:='p';
            name[nrofchars+2]:='b';
            name[nrofchars+3]:='s';
         end;
end;

begin
  repeat
     writeln('Inputfile ?');
     filecontrol(inputname,test,inp);
     if not test then writeln('Error - file does not exist: ',inputname);
  until test;
  writeln('Outputfile ?');
  filecontrol(outputname,test,outp);
  writeln;
  writeln('Inputfile: ',inputname);
  writeln('Outputfile: ',outputname);
  filename(inputname,status);
  open(pcodein,'$$$fil',old);
  reset(pcodein);
```

```
   filename(outputname,status);
   open(pbsout,'$$$fil',new);
   rewrite(pbsout);
end(* fileinit *);


(* - - forwards - - - - - - - - - - - - - - - - - - - - - -*)


procedure compile( symset : tset ); forward;

procedure insymbol; forward;

procedure skip( symset : tset ); forward;

procedure createtxt(tx : ttxtype); forward;


(* - - error and checking - - - - - - - - - - - - - - - -*)


function check(betw,small,big:integer):boolean;
begin
   check:= (betw>=small) and (betw<=big);
end (* check *);


procedure error(e:terror);
begin
   errflag:=true;
   write('*** Error - ');
   case e of
      esp:    writeln(' internal: compiler stack overflow');
      ecase:  writeln('error in case statement');
      eorg: writeln('pbs-address out of range:',s[sp].valu);
      eif:   writeln('more than ',ifmax,' nested if');
      echki:   writeln('array index out of range');
      epcaddr:  writeln('internal: pccount out of range');
      eitem: writeln('internal: stack-element type');
      elab:   writeln('internal: error in label: l',arg2:5);
      el:   writeln('internal: label limit violated');
      efil:   writeln('file error');
      eproc:   writeln('internal: format of pcode ');
      etxt:   writeln('internal: text managment');
      eext:  writeln('procedure should not be ',
                             'declared external: ',pctext);
   end (* case *);
   writeln(pbsout,'error****');
end (* error *);


procedure chk(e:terror);
begin
   case e of
      esp:   if not check(sp,0,slimit) then error(esp);
      eorg:  if not check(s[sp].valu,0,1777) then
                 error(eorg);
      eif:   if not check(iflev,1,ifmax) then error (eif);
      echki:   if not check(s[sp].valu,arg1,arg2) then
                  error(echki);
```

```
      el:   if not check(arg1,1,llimit) then error(el);
   end (* case *);
end (* chk *);


(* - - file managment - - - - - - - - - - - - - - - - - - - *)


procedure pcget;
begin
   get(pcfile);
   pccount:=pccount+1;
end (* pcget *);


procedure pcskipto(addr : integer);
begin
   if not check(addr,1,pcmaxaddr) then error(epcaddr);
   if pccount>=(addr-1) then
   begin
      pccount:=0;
      reset(pcfile);
   end;
   while (pccount<addr-1) and (pccount<pcmaxaddr) do pcget;
   with pcfile↑ do
   begin
      nextinstsym:=pcsym;
      nextargsym:=asym;
      nextarg1:=a1;
      nextarg2:=a2;
   end;
end (* pcskipto *);


procedure pcscanner; (* "cleans" pcodeinfile and moves
                                 the result to pcfile *)

procedure scan1;  (* part 1 of pcscanner *)
var   i : integer;
      lastproc : integer; (* label of last .ent-instr *)

   procedure symput;
   begin
      with symfile↑ do
      begin
         pcsym:=instsym;
         asym:=argsym;
         a1:=arg1;
         a2:=arg2;
      end;
      put(symfile);
   end(* symput *);

   procedure l1proc; (* put  label into the array l *)
   begin
      chk(el);
      if argsym = lbl then symput;
      with l[arg1] do
      case argsym of
      intlbl :
          begin
             level:=arg2;
             ltype:=store;
          end;
   lbl : case ltype of
            nolab:  ltype:=prog;
            cup,cxp:     ;
            end(* case *);
      end (* case *);
   end (* l1proc *);

   procedure inline; (* returns next p-code line
                                           from pcodein *)
   var   i:integer;
         flg : boolean;
         ch:char;

      function found(str:tpctext):boolean;
      var   find:boolean;
      begin
         find:=false; i:=0;
         repeat
            i := i+1;
            find := str=itxt[i];
         until find or (i=numbinstr);
         instsym := isym[i];
         argsym := iarg[i];
         found := find;
      end;

      procedure prname( tx:tpctext; prc:tcxp;
                           prtype:tproctype);
      begin
         if (pctext=tx) and not flg then
         begin
            l[arg1].ltype:=cxp;
            l[arg1].name:=prc;
            l[arg1].proctype:=prtype;
            flg:=true;
         end;
      end(* prname *);

   begin
      for i:=1 to lghtinstr do pctext[i]:=' ';
      i:=0;
      while not eoln(pcodein) and (i<lghtinstr) do
      begin
         i:=i+1;
         read(pcodein,ch);
         pctext[i]:=ch;
         if (i=1) and (ch='L') then i:=lghtinstr;
      end;
      if not found(pctext) then
      begin
         readln(pcodein);
         inline;
      end
      else
      begin
         case argsym of
         int:
            begin
```

```
      readln(pcodein,arg1);
      arg2:=-1;
   end(* int *);
int2:
   begin
      read(pcodein,arg1);
      readln(pcodein,arg2);
   end(* int2 *);
intlbl:
   begin
      read(pcodein,arg1);
      ch:=' ';
      while ch()'L' do read(pcodein,ch);
      readln(pcodein,arg2);
   end (* intlbl *) ;
lbl:
   begin
      read(pcodein,arg1);
      read(pcodein,ch);
      case ch of
      ':':
         begin
            readln(pcodein);
            arg2:=-1;
         end(* : *);
      '=':
         begin
            readln(pcodein,arg2);
            argsym:=intlbl;
         end(* = *);
      end (* case *);
   end (* lbl *);
jlbl:
   begin
      repeat
      read(pcodein,ch);
      until ch='L';
      readln(pcodein,arg1);
      arg2:=-1;
   end (* jlbl *);
proc:
   begin
      read(pcodein,ch);
      repeat
      read(pcodein,ch)
      until ch()' ';
      if ch='L' then
      (*** internal procedure ***)
      begin
         readln(pcodein,arg1);
         lastproc:=arg1;
         l[arg1].ltype:=cup;
      end
      else
      (*** external procedure ***)
      (* the procedure name may
         not begin with an 'l' *)
      begin
         pctext[1]:=ch;
         for i:=2 to 5 do
         begin
            read(pcodein,ch);
            pctext[i]:=ch;
         end;
         readln(pcodein);
         readln(pcodein);
         read(pcodein,ch);
         if ch()'L' then error(eproc);
         readln(pcodein,arg1);
         flg:=false;
         prname('TIMER',timer,procproc);
         prname('ORIGI',origin,procproc);
         prname('EDGE ',edge,boolproc);
         prname('EXOR ',exor,boolproc);
         prname('PBSAD',pbsadof,intproc);
         prname('CLEAR',clearpbs,procproc);
         if not flg then error(eext);
      end;
   end(* proc *);
none:
   begin
      readln(pcodein);
      arg1:=-1;
      arg2:=-1;
   end(* none *);
name:
   begin
      repeat
      read(pcodein,ch);
      until ch()' ';
      if ch='W' then
      begin
         read(pcodein,ch);
         if ch='L' then arg1:=3
         else
         begin
            read(pcodein,ch);
            if ch='S'
            then arg1:=2
            else if ch='I' then arg1:=1
            else arg1:=8;
         end;
      end
      else if ch='R' then
      begin
         read(pcodein,ch);
         if ch='L'
         then arg1:=4
         else arg1:=5;
      end
      else if ch='Q' then arg1:=6
      else arg1:=7;
      readln(pcodein);
      arg2:=-1;
      argsym:=int;
   end(* name *);
txt:
   begin
      repeat
      read(pcodein,ch);
      until (ch='"') or (ch='''');
      case ch of
```

```
          '"':
             begin
                createtxt(tex);
                for i:=1 to maxtxt do
                begin
                   read(pcodein,ch);
                   txtpool.last↑.t↑[i]:=ch;
                end;
             end(* " *);
          '''':
             begin
                createtxt(chr);
                read(pcodein,txtpool.last↑.c↑);
             end(* ' *);
          end(* case *);
          readln(pcodein);
          argsym:=int;
          arg1:=tcount;
          arg2:=-1;
       end(* txt *);
    end (* case *);
  end;
end (* inline *);

begin (* scan1 begins here *)
  rewrite(symfile);
  for i:=1 to 5 do readln(pcodein);
          (* ignore the first five lines of pcodein *)
  repeat
    inline;
    if instsym in [ldobsym,srobsym,laosym,ixasym,sroisym,
                   ldoisym,cupsym,cxpsym,indbsym,incasym,
                   movsym,indisym,ldoasym]
      then arg1:=arg1 div 2;
    if instsym in [ldasym,lodasym,strbsym,strisym,strasym,
                   lodisym,lodbsym]
    then arg2:= arg2 div 2 -5;
    if(instsym in [strbsym,strisym,strasym]) and (arg2=-5)
      then arg2:=0;
    if instsym in [ldobsym,srobsym,laosym,sroisym,
                   ldoisym,ldoasym] then arg1:= arg1-5;
    if (instsym=lsym) and (argsym=intlbl) then
      arg2:= arg2 div 2 -5;
    if instsym=lsym then l1proc
    else if instsym <> nosym then symput;
    if instsym=retbsym then l[lastproc].proctype:=boolproc
    else if instsym=retisym then
      l[lastproc].proctype:=intproc
    else if instsym=retpsym then
      l[lastproc].proctype:=procproc;
  until instsym = endsym;
end(* scan1 *);


procedure scan2;

type  pscannode=↑scannode;
      scannode=record
         instr: tinstr;
         atype: targ;
         a1,a2: integer;
           next: pscannode
        end;
var   fst,lst: pscannode;
      freescannodes: pscannode;
      labels: array[0..ifmax] of integer;
      scansp: integer;
      boolexpr: boolean;
      mstminuscup: integer;


  procedure symget;
  begin
    get(symfile);
    with symfile↑ do
    begin
       instsym:=pcsym;
       argsym:=asym;
       arg1:=a1;
       arg2:=a2;
    end;
  end (* symget *);

  procedure pcput(pcs:tinstr; as:targ; pa1,pa2:integer);
  begin
    if pcs=pcodesym then
       pcaddr:=pccount+1
    else
    with pcfile↑ do
    begin
       pcsym:=pcs;
       asym:=as;
       a1:=pa1;
       a2:=pa2;
       put(pcfile);
       pccount:=pccount+1;
    end;
  end (* pcput *);


  function newscannode: pscannode;
  var   temp: pscannode;
  begin
    if freescannodes=nil then
       new(temp)
    else
    begin
       temp:=freescannodes;
       freescannodes:=freescannodes↑.next;
    end;
    newscannode:=temp;
  end (* newscannode *);


  procedure disposescannode(garbage: pscannode);
  begin
    garbage↑.next:=freescannodes;
    freescannodes:=garbage;
  end (* disposescannode *);


  procedure retscannode(f: pscannode);
```

```
   begin
      if f<>nil then
      begin
         with f↑ do if instr<>nosym then
            pcput(instr,atype,a1,a2);
         retscannode(f↑.next);
         disposescannode(f);
      end;
   end (* retscannode *);


   procedure outscannode(f: pscannode);
   begin
      retscannode(f);
      fst:=nil;
      lst:=nil;
   end (* outscannode *);


   procedure intoscannode(ininstr:tinstr;inatype:targ;
                                         ina1,ina2: integer);
   var   tempscannode: pscannode;
   begin
      tempscannode:=newscannode;
      with tempscannode↑ do
      begin
         instr:=ininstr;
         atype:=inatype;
         a1:=ina1;
         a2:=ina2;
         next:=nil;
      end;
      if fst=nil then
      begin
         fst:=tempscannode;
         lst:=tempscannode;
      end
      else
      begin
         lst↑.next:=tempscannode;
         lst:=tempscannode;
      end;
   end (* intoscannode *);


begin  (* scan2 *)
   reset(symfile);
   rewrite(pcfile);
   labels[0]:=-1;
   boolexpr:=false;
   with symfile↑ do
   begin
      instsym:=pcsym;
      argsym:=asym;
      arg1:=a1;
      arg2:=a2;
   end;
   scansp:=0;
   freescannodes:=nil;
   fst:=nil;
   lst:=nil;
   repeat
      if lst <> nil then
         if (lst↑.instr=ujpsym) and (instsym<>lsym) then
            outscannode(fst);
      if instsym in [ldobsym,ldcbsym,lodbsym,indbsym] then
      begin
         boolexpr:=true;
         intoscannode(instsym,argsym,arg1,arg2);
      end
      else if instsym in [srobsym,stobsym,strbsym,sroisym,
                            strasym,strisym] then
      begin
         outscannode(fst);
         pcput(instsym,argsym,arg1,arg2);
         boolexpr:=false;
      end
      else if instsym=fjpsym then
      begin
         if boolexpr then
         begin
            pcput(ifsym,none,-1,-1);
            outscannode(fst);
            pcput(thensym,none,-1,-1);
            scansp:=scansp+1;
            if scansp>ifmax then error(eif);
            labels[scansp]:=arg1;
                boolexpr:=false;
         end
         else
         begin
            outscannode(fst);
            pcput(instsym,argsym,arg1,arg2);
         end;
      end
      else if instsym=lsym then
      begin
         if arg1=labels[scansp] then
         begin
            if lst<>nil then
               if (lst↑.instr=ujpsym) then
               begin
                  lst↑.instr:=nosym;
                  labels[scansp]:=lst↑.a1;
                  outscannode(fst);
                  pcput(elsesym,none,-1,-1);
               end
               else
               begin
                  scansp:=scansp-1;
                  outscannode(fst);
                  pcput(eifsym,none,-1,-1);
               end
            else
            begin
               scansp:=scansp-1;
               pcput(eifsym,none,-1,-1);
            end
         end
         else
         begin
```

```
         outscannode(fst);
         l[arg1].level:=pccount+1;
      end;
    end
    else if instsym=mstsym then
    begin
      mstminuscup:=1;
      intoscannode(instsym,argsym,arg1,arg2);
      repeat
         symget;
         if instsym=mstsym then
            mstminuscup:=mstminuscup+1
         else if instsym in [cupsym,cxpsym] then
            mstminuscup:=mstminuscup-1;
         intoscannode(instsym,argsym,arg1,arg2);
      until mstminuscup=0;
      case l[arg2].proctype of
         procproc: outscannode(fst);
         intproc:  ;
         boolproc: boolexpr:=true;
      end;
    end
    else if instsym in [entsym,retbsym,retisym,retpsym,
                        stpsym,xjpsym,ujcsym,cspsym] then
    begin
      outscannode(fst);
      pcput(instsym,argsym,arg1,arg2);
    end
    else
      intoscannode(instsym,argsym,arg1,arg2);
    symget;
  until instsym = endsym;
  intoscannode(instsym,argsym,arg1,arg2);
  outscannode(fst);
  pcmaxaddr:=pccount;
  pcskipto(pcaddr);
end (* scan2 *);

begin (* pcscanner *);
  scan1;
  scan2;
end (* pcscanner *);

procedure insymbol;(*returns next p-code line from pcfile*)
begin
  if nextinstsym<>endsym then pcget;
  instsym:=nextinstsym;
  argsym:=nextargsym;
  arg1:=nextarg1;
  arg2:=nextarg2;
  if nextinstsym<>endsym then
  with pcfile^ do
  begin
    nextinstsym:=pcsym;
    nextargsym:=asym;
    nextarg1:=a1;
    nextarg2:=a2;
  end (* with *);
end (* insymbol *);

(* - - code generator - - - - - - - - - - - - - - - - -  *)

procedure genpbs(nr:integer;arg:pbsrange);
begin
  if not wlnflgprr then
  begin
    writeln(pbsout);
    wlnflgprr:=true;
  end;
  write(pbsout,pbstxt[nr]);
  if pbsarg[nr] then write(pbsout,arg:10);
  writeln(pbsout);
end (* genpbs *);


(* - - octal<>decimal - - - - - - - - - - - - - - - - -  *)


function octal(d:integer): integer;
                              (* converts decimal to octal *)
begin
  if d=0 then octal:=0
  else octal:= d mod 8 + 10*octal(d div 8);
end (* octal *);


function decimal(o:integer): integer;
                              (* converts octal to decimal *)
begin
  if o=0 then decimal:=0
  else decimal:= o mod 10 + 8*decimal(o div 10);
end (* decimal *);


procedure octinc(var o:integer);
begin
  o:=octal(decimal(o)+1);
end (* octinc *);


procedure octdec(var o:integer);
begin
  o:=octal(decimal(o)-1);
end (* octdec *);


(* - - pbsaddress-table managment - - - - - - - - - - -  *)


function getaddress(typewanted: taddr):pbsrange;

var found : boolean;

begin
  found:=false;
  while not found and (ap<>rp) do
  if (pbsaddr[ap]=typewanted) or (typewanted=usedbefore)
     and (pbsaddr[ap]=free) then found:=true
  else if ap=pahigh
       then ap:=aplowbound
       else ap:=ap+1;
```

```
   if ap=rp then
   if aplowbound = pamidlow then
   begin
      writeln('free pbsaddresses between',
              '0 and 777 (octal) are used');
      aplowbound:=palow+1;
   end
   else
   begin
      writeln('no more pbs-addresses available!');
      getaddress:=-1;
   end
   else
   begin
      pbsaddr[ap]:=used;
      getaddress:=octal(ap);
   end;
   if ap=aplowbound then rp:=pahigh else rp:=ap-1;
end(* getaddress *);

procedure leaveaddress(octaddr : pbsrange);
begin
   if returnaddresses then
   pbsaddr[decimal(octaddr)]:=usedbefore;
end(* leaveaddress *);

procedure assignaddress(selem: integer;typewanted: taddr);
begin
   with s[selem] do
   if itype<>addr then
   begin
      itype:=addr;
      address:=getaddress(typewanted);
      neg:=false;
   end;
end(* assignaddress *);


procedure leavelist;
begin
   with leavepool do
   begin
      while rp<>ap do
      begin
         leaveaddress(rp↑.val);
         rp:=rp↑.next;
      end(* while *);
      rp:=fp;
      ap:=fp;
   end(* with *);
end(* leavelist *);

procedure pushleave;
begin
   with leavepool do
   begin
      ap↑.val:=s[sp].address;
      if ap↑.next=nil then
      begin
         new(ap↑.next);
         ap↑.next↑.next:=nil;
      end;
      ap:=ap↑.next;
   end(* with *);
end(* pushleave *);


(* - - pool managment - - - - - - - - - - - - - - - - - *)

(* the procedures intopool,outofpool and disposenodes only
works on stack elements of type 'expr' *)

procedure intopool(p : nodep);
begin
   with p↑ do .
   begin
      next:=nil;
      nodeinstr:=-1;
      nodearg:=-1;
   end(* with *);
   with pool do
   if first = nil then
   begin
      first:=p;
      last:=p;
   end
   else
   begin
      last↑.next:=p;
      last:=p;
   end;
end (* intopool *);

function outofpool : nodep;
var   p : nodep;

   procedure createnodes(nrofnodes : integer);
   var   p : nodep;
   begin
      new(p);
      intopool(p);
      if nrofnodes > 1 then createnodes(nrofnodes-1);
   end(* createnodes *);

begin
   with pool do
   begin
      if first = nil then createnodes(10);
      p:=first;
      first:=first↑.next;
      p↑.next:=nil;
      outofpool:=p;
   end(* with *);
end(* outofpool *);

procedure disposenodes(p : nodep; snr : integer);

   procedure retnode(p1 : nodep);
   begin
      if p1↑.next<>nil then retnode(p1↑.next);
      if p1↑.nodeinstr in [5,6] then
```

```
         leaveaddress(p1↑.nodearg);
      intopool(p1);
   end(* retnode *);

begin
   retnode(p);
   s[snr].first:=nil;
   s[snr].last:=nil;
end(* disposenodes *);



(* - - stack managment - - - - - - - - - - - - - - - - - - *)



procedure intoexpr(var fst,lst:nodep;i:integer;a:pbsrange);
var temp : nodep;
begin
   temp:=outofpool;
   temp↑.nodeinstr:=i;
   temp↑.nodearg:=a;
   if fst=nil then
   begin
      fst:=temp;
      lst:=temp;
   end
   else
   begin
      temp↑.next:=lst↑.next;
      lst↑.next:=temp;
      lst:=temp;
   end;
end (* intoexpr *);


procedure precedefirst(snr,instr: integer;addr: pbsrange);
(* this procedure assumes first <> nil *)
var   temp : nodep;
begin
   temp:= outofpool;
   with temp↑ do
   begin
      nodeinstr:= instr;
      nodearg:= addr;
      next:= s[snr].first;
   end;
   s[snr].first:= temp;
end; (* precedefirst *)


procedure outnode(fst:nodep);
begin
   if fst<>nil then
   begin
      genpbs(fst↑.nodeinstr,fst↑.nodearg);
      outnode(fst↑.next);
   end;
end (* outnode *);


procedure sesearch(var sepoint : nodep;f,l : nodep);
var   hp : nodep; (* help pointer *)
begin
   sepoint:=nil;
   hp:=f;
   while hp<>l do
   if hp↑.nodeinstr in [5,6] then
   begin
      sepoint:=hp;
      hp:=hp↑.next;
   end
   else hp:=hp↑.next;
end(* sesearch *);


procedure swapitem(var a,b:titem);
var   temp : titem;
begin
   temp:=a;
   a:=b;
   b:=temp;
end (* swapitem *);


function searchaddr(lev,disp : integer): integer;

   function saddr(p,l : integer):integer;
   begin
      if l>0
      then saddr:=saddr(s[p].statlink,l-1)
      else saddr:=p;
   end(* saddr *);
begin
   searchaddr:=saddr(mp,lev)+disp;
end (* searchaddr *);


procedure decstack;
begin
   s[sp].itype:=noitem;
   sp:=sp-1;
end(* decstack *);

procedure incstack;
begin
   sp:=sp+1;
   chk(esp);
end(* incstack *);



(* - - text managment - - - - - - - - - - - - - - - - - - *)


procedure createtxt(*(tx : ttxtype )*);
begin
   tcount:=tcount+1;
   with txtpool do
   begin
      if last = nil then
```

```
      begin
         new(first);
         last:=first;
      end
      else
      begin
         new(last↑.next);
         last:=last↑.next;
      end;
      last↑.txtype:=tx;
      case tx of
      chr:   new(last↑.c);
      tex:   new(last↑.t);
      end(* case *);
      last↑.tnr:=tcount;
   end(* with *);
end(* createtxt *);

function searchtxt( nr : integer): ptxtnode;
var   i : integer;
      found : boolean;
      p : ptxtnode;
begin
   found:=false;
   if nr=tcount then searchtxt:=txtpool.last
   else
   begin
      i:=1;
      p:=txtpool.first;
      while (i(tcount) and not found do
      if i=nr then found:=true
      else
      begin
         i:=i+1;
         p:=p↑.next;
      end;
      if not found then error(etxt)
      else searchtxt:=p;
   end;
end(* searchtxt *);


(* read/readln and write/writeln:

   only read/readln and write/writeln on integers are
   implemented

   write/writeln on strings between quotes ('')
   are implemented *)

procedure cspproc;
var   ptxt : ptxtnode;
      i,c : integer;

   function cdigits( intnr : integer ) : integer;
   begin
      if intnr()0
      then cdigits:=cdigits(intnr div 10)+1
      else cdigits:=0;
   end(* cdigits *);
```

```
begin
   case arg1 of
   1:   (* wri *)
   begin
      with s[s[sp].valu] do
      if itype()fil then error(eitem)
      else
      begin
         c:=cdigits(s[sp-2].valu);
         if s[sp-2].valu(0 then c:=c+1;
         case ftype of
         outfile:
         begin
            for i:=1 to s[sp-1].valu-c do
            write(output,' ');
            write(output,s[sp-2].valu:c);
         end(* outfile *);
         pbsfile:
         begin
            if wlnflgprr then write(pbsout,'*C* ');
            for i:=1 to s[sp-1].valu-c do
            write(pbsout,' ');
            write(pbsout,s[sp-2].valu:c);
            wlnflgprr:=false;
         end(* pbsfile *);
         infile:   error(efil);
         end(* case *);
      end(* with *);
      for i:=1 to 3 do decstack;
   end(* wri *);
   2:   (* wrs *)
   begin
      ptxt:=searchtxt(s[sp-3].valu);
      with s[s[sp].valu] do
      if itype()fil then error(eitem)
      else
      begin
         if wlnflgprr and (ftype = pbsfile) then
            write(pbsout,'*C* ');
         for i:=1 to s[sp-2].valu do
         case ftype of
         outfile: write(output,ptxt↑.t↑[i]);
         pbsfile: write(pbsout,ptxt↑.t↑[i]);
         infile: error(efil);
         end(* case for *);
         if ftype=pbsfile then wlnflgprr:=false;
      end(* with *);
      for i:=1 to 4 do decstack;
   end(* wrs *);
   3:   (* wln *)
   begin
      with s[s[sp].valu] do
      if itype()fil then error(eitem)
      else
      case ftype of
      infile: error(efil);
      outfile: writeln(output);
      pbsfile: begin writeln(pbsout);wlnflgprr:=true end;
      end(* case with *);
      decstack;
   end(* wln *);
```

```
4:   (* rln *)
begin
   with s[s[sp].valu] do
   if itype<>fil then error(eitem)
   else
   case ftype of
   pbsfile,outfile: error(efil);
   infile: readln(input);
   end(* case with *);
   decstack;
end(* rln *);
5:   (* rdi *)
begin
   with s[s[sp].valu] do
   if itype<>fil then error(eitem)
   else
   case ftype of
   pbsfile,outfile: error(efil);
   infile:   with s[s[sp-1].valu] do
      begin
         itype:=val;
         read(input,valu);
         neg:=false;
      end(* infile *);
   end(* case with *);
   decstack;
   decstack;
end(* rdi *);
6:   (* opn *)
begin
   ptxt:=searchtxt(s[sp-2].valu);
   with s[s[sp].valu] do
   begin
      itype:=fil;
      with ptxt↑ do
      if t↑=cout then ftype:=outfile
      else if t↑=cprd then ftype:=infile
      else if t↑=cprr then ftype:=pbsfile
      else error(efil);
      neg:=false;
   end(* with *);
   for i:=1 to 3 do decstack;
end(* opn *);
7:   (* cls *)
begin
   decstack;
end(* cls *);
8:   (* wrc *)
begin
   ptxt:=searchtxt(s[sp-2].valu);
   with s[s[sp].valu] do
   if itype<>fil then error(eitem)
   else
   begin
      if wlnflgprr and (ftype=pbsfile) then
         write(pbsout,'*C* ');
      case ftype of
         pbsfile: write(pbsout,ptxt↑.c↑);
         outfile: write(output,ptxt↑.c↑);
         infile: error(efil);
      end(* case *);
         if ftype=pbsfile then wlnflgprr:=false;
      end(* with *);
      for i:=1 to 3 do decstack;
   end(* wrc *);
   end(* case *);
end(* cspproc *);


(* - - system procedures - - - - - - - - - - - - - - - *)

procedure gencnt(time : integer);
begin
   genpbs(10+time mod 2,getaddress(free));
   if time div 2 > 0 then gencnt(time div 2);
end(* gencnt *);

procedure timproc;
var   pbssp : pbsrange;
      sp3nl,sp4nl : boolean;
begin
   sp3nl:=true;
   sp4nl:=true;
   with s[sp-4] do
   if itype=expr then
   begin
      pbssp:=getaddress(usedbefore);
      outnode(first);
      if neg then genpbs(6,pbssp)   (*sen*)
             else genpbs(5,pbssp); (*se*)
      disposenodes(first,sp-4);
      itype:=addr;
      address:=pbssp;
      neg:=false;
      sp4nl:=false;
   end;
   with s[sp-3] do
   if itype=expr then
   begin
      pbssp:=getaddress(usedbefore);
      outnode(first);
      if neg then genpbs(6,pbssp)
             else genpbs(5,pbssp);
      disposenodes(first,sp-3);
      itype:=addr;
      address:=pbssp;
      neg:=false;
      sp3nl:=false;
   end;
   if s[sp-4].neg   then genpbs(1,s[sp-4].address)
        else genpbs(2,s[sp-4].address);
   assignaddress(s[sp-2].valu,free);
   genpbs(3,s[s[sp-2].valu].address);
   genpbs(14,getaddress(free));
   if s[sp-3].itype=addr then
   if s[sp-3].neg   then genpbs(1,s[sp-3].address)
        else genpbs(2,s[sp-3].address);
   gencnt(s[sp].valu);
   assignaddress(s[sp-1].valu,free);
   genpbs(13,s[s[sp-1].valu].address);
   if sp4nl then s[sp-4].itype:=noitem;
```

```
    s[sp-4].neg:=false;
    if sp3nl then s[sp-3].itype:=noitem;
    s[sp-3].neg:=false;
    instsym:=retpsym;
    argsym:=none;
    arg1:=-1;
    arg2:=-1;
 end(* timproc *);

 procedure orgproc;
 begin
    chk(eorg);
    with s[s[sp-1].valu] do
    if itype=addr then
    begin
       if address()s[sp].valu then
       begin
          writeln('variabel was already assigned a pbsaddr');
          writeln('new pbsaddr:',s[sp].valu);
          writeln('old pbsaddr:',address);
          writeln('the new pbsaddress is assumed');
          leaveaddress(address);
          address:=s[sp].valu;
       end;
    end
    else
    begin
       if itype()noitem then error(eitem);
       itype:=addr;
       neg:=false;
       address:=s[sp].valu;
    end;
    if (pbsaddr[decimal(s[sp].valu)] = used)
       and (s[sp].valu()1000) and (s[sp].valu()1001) then
    writeln('this pbsaddress was already used: ',
                         s[s[sp-1].valu].address:5);
    pbsaddr[decimal(s[sp].valu)]:=used;
    instsym:=retpsym;
    argsym:=none;
    arg1:=-1;
    arg2:=-1;
 end(* orgproc *);


 procedure exedgeproc( instrnr : integer );
 (*  only 'ex', not 'exn', is implemented *)
 (* 'exn' is implemented in pbsopt *)

 var    pbssp : pbsrange;
        p,parin : nodep;

    function orsearch(selem:integer;var remp:nodep):boolean;

    var    p : nodep;
           orfound : boolean;

    begin
       orfound:=false;
       with s[selem] do
       if itype()expr then error(eitem)
       else
       begin
          remp:=first;
          p:=first;
          while p()nil do  (* last↑.next is always nil *)
          begin
             if p↑.nodeinstr in [3,4]
             then orfound:=true
             else if p↑.nodeinstr in [5,6] then
             begin
                remp:=p;
                orfound:=false;
             end;
             p:=p↑.next;
          end;
       end(* with *);
       orsearch:=orfound;
    end(* orsearch *);

 begin
    pbssp:=getaddress(usedbefore);
    with s[sp-1] do
    if itype=expr then
    begin
       if ((instrnr=15) or (instrnr=14) and not neg)
            and orsearch(sp-1,parin) then
       begin
          if (parin = first) and (first () nil)
          then precedefirst(sp-1,16,0)
          else intoexpr(first,parin,16,0);
          intoexpr(first,last,17,0);
       end;
       if neg then
       begin
          intoexpr(first,last,6,pbssp);
          intoexpr(first,last,1,pbssp);
          pbssp:=getaddress(usedbefore);
       end;
       outnode(first);
    end
    else
    if itype=addr then
       if neg then genpbs(2,address)
       else genpbs(1,address)
    else error(eitem);
    assignaddress(s[sp].valu,free);
    if s[sp-1].itype=expr then
       disposenodes(s[sp-1].first,sp-1);
    s[sp-1].itype:=noitem;
    s[sp-1].neg:=false;
    genpbs(instrnr,s[s[sp].valu].address);
    genpbs(5,pbssp);
    s[mp].return:=pbssp;
    instsym:=retbsym;
    argsym:=none;
    arg1:=-1;
    arg2:=-1;
 end(* exedgeproc *);


 procedure paproc;
```

```
(*pushes the pbsaddress of a boolean variable on stack *)
begin
   with s[s[sp].valu] do
   if itype()addr
   then s[mp].return:=-1
   else s[mp].return:=address;
   instsym:=retisym;
   argsym:=none;
   arg1:=-1;
   arg2:=-1;
end(* paproc *);


procedure clearproc;
var   pbssp : pbsrange;
begin
   case s[sp].itype of
   val:   error(eitem);
   addr:
      begin
         if s[sp].neg
         then genpbs(2,s[sp].address)
         else genpbs(1,s[sp].address);
         genpbs(18,0);
      end(* addr *);
   expr:
      begin
         outnode(s[sp].first);
         if s[sp].neg then
         begin
            pbssp:=getaddress(usedbefore);
            genpbs(6,pbssp);
            genpbs(1,pbssp);
         end;
         genpbs(18,0);
         disposenodes(s[sp].first,sp);
         leaveaddress(pbssp);
      end(* expr *);
   end(* case *);
   s[sp].itype:=noitem;
   instsym:=retpsym;
   argsym:=none;
   arg1:=-1;
   arg2:=-1;
end(* clearproc *);


(* - - load - - - - - - - - - - - - - - - - - - - - - - - - *)


procedure ldobproc; (*load boolean from base level address*)
begin
   assignaddress(arg1,free);
   incstack;
   s[sp].itype:=addr;
   s[sp].address:=s[arg1].address;
   s[sp].neg:=false;
end (* ldobproc *);


procedure indbproc;   (* indexed fetch of boolean *)
begin
   arg1:=s[sp].valu+arg1;
   decstack;
   ldobproc;
end (* indbproc *);


procedure ldoproc; (*load integer from base level address*)
begin
   incstack;
   with s[sp] do
   begin
      itype:=val;
      valu:=s[arg1].valu;
   end;
end (* ldoproc *);


procedure indiproc;   (* indexed fetch of integer *)
begin
   arg1:=s[sp].valu+arg1;
   decstack;
   ldoproc;
end(* indiproc *);

procedure ldcproc;   (* load constant *)
begin
   incstack;
   s[sp].itype:=val;
   s[sp].valu:=arg1;
   s[sp].neg:=false;
end (* ldcproc *);


procedure lodproc; (* load integer from relative address *)
begin
   incstack;
   with s[sp] do
   begin
      itype:=val;
      valu:=s[searchaddr(arg1,arg2)].valu;
      neg:=false;
   end (* with *);
end (* lodproc *);


procedure lodbproc; (* load boolean from relative address *)
begin
   assignaddress(searchaddr(arg1,arg2),free);
   incstack;
   with s[sp] do
   begin
      itype:=addr;
      address:=s[searchaddr(arg1,arg2)].address;
      neg:=false;
   end (* with *);
end (* lodbproc *);


procedure ldaproc;   (* load address direct *)
begin
```

```
   incstack;
   with s[sp] do
   begin
      itype:=val;
      valu:=searchaddr(arg1,arg2);
      neg:=false;
   end (* with *);
end (* ldaproc *);

(* - - store - - - - - - - - - - - - - - - - - - - - - - *)


procedure sroiproc; (*store integer at base level address*)
begin
   with s[arg1] do
   begin
      itype:=val;
      valu:=s[sp].valu;
   end;
   decstack;
end (* sroiproc *);


procedure srobproc; (*store boolean at base level address*)
begin
   assignaddress(arg1,free);
   case s[sp].itype of
   val:
      if s[sp].valu = 0 then
         genpbs(6,s[arg1].address) (*sen*)
      else genpbs(5,s[arg1].address); (*se*)
   expr :
      begin
         outnode(s[sp].first);
         if s[sp].neg then genpbs(6,s[arg1].address) (*sen*)
         else genpbs(5,s[arg1].address); (*se*)
         disposenodes(s[sp].first,sp);
      end;
   addr :
      begin
         if s[sp].neg then
            genpbs(2,s[sp].address). (*adn*)
         else
            genpbs(1,s[sp].address); (*ad*)
         genpbs(5,s[arg1].address); (*se*)
      end;
   end (* case *);
   decstack;
   if semaphor=0 then leavelist;
end (* srobproc *);


procedure stobproc;   (* store boolean *)
begin
   swapitem(s[sp],s[sp-1]);
   arg1:=s[sp].valu;
   decstack;
   srobproc;
end (* stobproc *);
```

```
procedure stoiproc;   (* store integer *)
begin
   swapitem(s[sp],s[sp-1]);
   arg1:=s[sp].valu;
   decstack;
   sroiproc;
end(* stoiproc *);

procedure strbproc; (*store boolean at relative address*)
begin
   if (arg1=0) and (arg2=0) then
   begin
   if s[mp].return=-1 then
      s[mp].return:=getaddress(usedbefore);
      case s[sp].itype of
      val:
         begin
            if s[sp].valu=0 then genpbs(6,s[mp].return)
            else genpbs(5,s[mp].return);
         end(* val *);
      addr:
         begin
            if s[sp].neg then genpbs(2,s[sp].address)
            else genpbs(1,s[sp].address);
            genpbs(5,s[mp].return);
         end(* addr *);
      expr:
         begin
            outnode(s[sp].first);
            if s[sp].neg then genpbs(6,s[mp].return)
            else genpbs(5,s[mp].return);
            disposenodes(s[sp].first,sp);
         end(* expr*);
      end(* case *);
      decstack;
   end
   else
   begin
      arg1:=searchaddr(arg1,arg2);
      srobproc;
   end;
end (* strbproc *);


procedure strproc;   (* store integer at relative address *)
begin
   if (arg1=0) and (arg2=0)
   then s[mp].return:=s[sp].valu
   else
   begin
      with s[searchaddr(arg1,arg2)] do
      begin
         itype:=val;
         valu:=s[sp].valu;
         neg:=s[sp].neg;
      end (* with *);
   end;
   decstack;
end (* strproc *);
```

```
(* - - integer/address - - - - - - - - - - - - - - - - - *)

procedure adiproc;
begin
   s[sp-1].valu:=s[sp-1].valu+s[sp].valu;
   decstack;
end (* adiproc *);

procedure sbiproc;
begin
   s[sp-1].valu:=s[sp-1].valu-s[sp].valu;
   decstack;
end (* sbiproc *);

procedure mpiproc;
begin
   s[sp-1].valu:=s[sp-1].valu*s[sp].valu;
   decstack;
end (* mpiproc *);

procedure dviproc;
begin
   s[sp-1].valu:=s[sp-1].valu div s[sp].valu;
   decstack;
end (* dviproc *);

procedure modproc;
begin
   s[sp-1].valu:=s[sp-1].valu mod s[sp].valu;
   decstack;
end (* modproc *);

procedure ixaproc;   (* compute indexed address *)
begin
   s[sp-1].valu:=s[sp].valu*arg1+s[sp-1].valu;
   decstack;
end (* ixaproc *);

procedure deciproc;  (* decrease stack top value by arg1 *)
begin
   s[sp].valu:=s[sp].valu-arg1;
end (* deciproc *);

procedure incproc;   (* increase stack top value by arg1 *)
begin
   s[sp].valu:=s[sp].valu+arg1;
end (* incproc *);

procedure equiproc;
begin
 if s[sp-1].valu=s[sp].valu then
    s[sp-1].valu:=1 else s[sp-1].valu:=0;
 decstack;
end (* equiproc *);

procedure grtiproc;
begin
   if s[sp-1].valu>s[sp].valu then
      s[sp-1].valu:=1 else s[sp-1].valu:=0;
   decstack;
end (* grtiproc *);

procedure geqiproc;
begin
   if s[sp-1].valu>=s[sp].valu then
      s[sp-1].valu:=1 else s[sp-1].valu:=0;
   decstack;
end (* geqiproc *);

procedure lesiproc;
begin
   if s[sp-1].valu<s[sp].valu then
      s[sp-1].valu:=1 else s[sp-1].valu:=0;
   decstack;
end (* lesiproc *);

procedure leqiproc;
begin
   if s[sp-1].valu<=s[sp].valu then
      s[sp-1].valu:=1 else s[sp-1].valu:=0;
   decstack;
end (* leqiproc *);

procedure neqiproc;
begin
   if s[sp-1].valu<>s[sp].valu then
      s[sp-1].valu:=1 else s[sp-1].valu:=0;
   decstack;
end (* neqiproc *);

procedure ngiproc;
begin
   s[sp].valu:=-s[sp].valu;
end(* ngiproc *);

(* - - boolean - - - - - - - - - - - - - - - - - - - - - *)

procedure andproc; (* perform boolean and *)
var   st1,st2 : titem;
      pse : nodep;
      i : integer;
      pbssp : pbsrange;
begin
   st1:=s[sp];
   st2:=s[sp-1];
   if (st1.itype = addr) and (st2.itype = addr) then
```

```
begin
   with s[sp-1] do
   begin
      itype:=expr;
      neg:=false;
      first:=nil;
      last:=nil;
      andlegal:=both;
      orlegal:=both;
      if st2.neg
      then intoexpr(first,last,2,st2.address)
      else intoexpr(first,last,1,st2.address);
      if st1.neg
      then intoexpr(first,last,2,st1.address)
      else intoexpr(first,last,1,st1.address);
   end(* with *);
   decstack;
end(* addr,addr *)
else if (st1.itype = val) and (st2.itype = val) then
begin
   if (st1.valu=1) and (st2.valu=1)
   then s[sp-1].valu:=1
   else s[sp-1].valu:=0;
   decstack;
end (* val,val *)
else if (st1.itype = expr) and (st2.itype = expr) then
begin
   for i:=0 to 1 do
   with s[sp-1] do
   if neg then
   begin
      pbssp:=getaddress(usedbefore);
      intoexpr(first,last,6,pbssp);
      intoexpr(first,last,1,pbssp);
      andlegal:=lastend;
      orlegal:=lastend;
      neg:=false;
   end(* if-with-for *);
   case st1.andlegal of
   both:
      case st2.andlegal of
      both,lastend:
         begin
            with s[sp-1] do
            begin
               st2.last↑.next:=st1.first;
               last:=st1.last;
            end(* with *);
            decstack;
         end(* both,lastend *);
      noend:
         begin
            pbssp:=getaddress(usedbefore);
            intoexpr(st2.first,st2.last,5,pbssp);
            intoexpr(st2.first,st2.last,1,pbssp);
            st2.last↑.next:=st1.first;
            with s[sp-1] do
            begin
               last:=st1.last;
               andlegal:=lastend;
               orlegal:=lastend;
            end;
            decstack;
         end(* noend *);
      end(* case *);
   lastend:
      case st2.andlegal of
      both:
         begin
            swapitem(s[sp],s[sp-1]);
            andproc;
         end(* both *);
      lastend:
         begin
            sesearch(pse,st1.first,st1.last);
            st2.last↑.next:=pse↑.next;
            pse↑.next:=st2.first;
            with s[sp-1] do
            begin
               first:=st1.first;
               last:=st1.last;
            end(* with *);
            decstack;
         end(* lastend *);
      noend:
         begin
            pbssp:=getaddress(usedbefore);
            intoexpr(st2.first,st2.last,5,pbssp);
            intoexpr(st1.first,st1.last,1,pbssp);
            st2.last↑.next:=st1.first;
            with s[sp-1] do
            begin
               last:=st1.last;
               andlegal:=lastend;
               orlegal:=lastend;
            end(* with *);
            decstack;
         end;
      end(* case *);
   noend:
      case st2.andlegal of
      both,lastend:
         begin
            swapitem(s[sp],s[sp-1]);
            andproc;
         end(* both,lastend *);
      noend:
         begin
            pbssp:=getaddress(usedbefore);
            intoexpr(st1.first,st1.last,5,pbssp);
            intoexpr(st1.first,st1.last,1,pbssp);
            pbssp:=getaddress(usedbefore);
            intoexpr(st2.first,st2.last,5,pbssp);
            intoexpr(st1.first,st1.last,1,pbssp);
            st2.last↑.next:=st1.first;
            with s[sp-1] do
            begin
               last:=st1.last;
               andlegal:=lastend;
               orlegal:=lastend;
            end(* with *);
            decstack;
```

```
               end(* noend *);
            end(* case *);
         end(* case *);
      end(* expr,expr *)
      else if (st2.itype = addr) and (st1.itype = expr) then
      begin
         swapitem(s[sp],s[sp-1]);
         andproc;
      end(* addr,expr *)
      else if (st2.itype = expr) and (st1.itype = addr) then
      begin
         with s[sp-1] do
         begin
            if neg then
            begin
               pbssp:=getaddress(usedbefore);
               intoexpr(first,last,6,pbssp);
               intoexpr(first,last,1,pbssp);
               andlegal:=lastend;
               orlegal:=lastend;
               neg:=false;
            end(* if *);
            if andlegal = noend then
            begin
               pbssp:=getaddress(usedbefore);
               intoexpr(first,last,5,pbssp);
               intoexpr(first,last,1,pbssp);
               andlegal:=lastend;
               orlegal:=lastend;
            end;
            if st1.neg
            then intoexpr(first,last,2,st1.address)
            else intoexpr(first,last,1,st1.address);
         end(* with *);
         decstack;
      end(* expr,addr *)
      else if ((st1.itype=addr) or (st1.itype=expr))
         and (st2.itype=val) then
      begin
         if st2.valu=1 then swapitem(s[sp],s[sp-1]);
         if s[sp].itype=expr then disposenodes(s[sp].first,sp);
         decstack;
      end (* addr,val  expr,val *)
      else if (st1.itype=val)
         and ((st2.itype=addr) or (st2.itype=expr)) then
      begin
         if st1.valu=0 then swapitem(s[sp],s[sp-1]);
         if s[sp].itype=expr then disposenodes(s[sp].first,sp);
         decstack;
      end (* val,addr  val,expr *);
end (* andproc *);


procedure iorproc;   (* perform boolean (inclusive) or *)
var   st1,st2 : titem;
      pse : nodep;
      i : integer;
      pbssp : pbsrange;
begin
   st1:=s[sp];
   st2:=s[sp-1];
      if (st1.itype=addr) and (st2.itype=addr) then
      begin
         with s[sp-1] do
         begin
            itype:=expr;
            neg:=false;
            first:=nil;
            last:=nil;
            orlegal:=both;
            andlegal:=noend;
            if st2.neg then
               intoexpr(first,last,2,st2.address)
            else
               intoexpr(first,last,1,st2.address);
            if st1.neg then
               intoexpr(first,last,4,st1.address)
            else
               intoexpr(first,last,3,st1.address);
         end (* with *);
         decstack;
      end (* addr,addr *)
      else if (st1.itype=val) and (st2.itype=val) then
      begin
         if (st1.valu=1) or (st2.valu=1) then
            s[sp-1].valu:=1
         else
            s[sp-1].valu:=0;
         decstack;
      end (* val,val *)
      else if (st1.itype=expr) and (st2.itype=expr) then
      begin
         for i:=0 to 1 do
         with s[sp-i] do
         if neg then
         begin
            pbssp:=getaddress(usedbefore);
            intoexpr(first,last,6,pbssp);
            intoexpr(first,last,1,pbssp);
            andlegal:=lastend;
            orlegal:=lastend;
            neg:=false;
         end(* if-with-for *);
         case st1.orlegal of
         both:
            case st2.orlegal of
            both,lastend:
               begin
                  with s[sp-1] do
                  begin
                     st2.last↑.next:=st1.first;
                     st1.first↑.nodeinstr:=
                                    st1.first↑.nodeinstr+2;
                     last:=st1.last;
                     andlegal:=noend;
                  end(* with *);
                  decstack;
               end(* both,lastend *);
            noend:   ;
            end(* case *);
         lastend:
            case st2.orlegal of
```

```
      both:
         begin
            swapitem(s[sp],s[sp-1]);
            iorproc;
         end(* both *);
      lastend:
         begin
            sesearch(pse,st1.first,st1.last);
            st2.last↑.next:=pse↑.next;
            pse↑.next↑.nodeinstr:=pse↑.next↑.nodeinstr+2;
            pse↑.next:=st2.first;
            with s[sp-1] do
            begin
               first:=st1.first;
               last:=st1.last;
               andlegal:=noend;
            end(* with *);
            decstack;
         end(* lastend *);
      noend:   ;
      end(* case *);
   noend:   ;
   end(* case *);
end (* expr,expr *)
else if (st1.itype=expr) and (st2.itype=addr) then
begin
   swapitem(s[sp],s[sp-1]);
   iorproc;
end (* expr,addr *)
else if (st1.itype=addr) and (st2.itype=expr) then
begin
   with s[sp-1] do
   begin
      if neg then
      begin
         pbssp:=getaddress(usedbefore);
         intoexpr(first,last,6,pbssp);
         intoexpr(first,last,1,pbssp);
         andlegal:=lastend;
         orlegal:=lastend;
         neg:=false;
      end(* if *);
      if st1.neg
      then intoexpr(first,last,4,st1.address)
      else intoexpr(first,last,3,st1.address);
      andlegal:=noend;
   end(* with *);
   decstack;
end (* addr,expr *)
else if ((st1.itype=addr) or (st1.itype=expr))
   and (st2.itype=val) then
begin
   if st2.valu=0 then swapitem(s[sp],s[sp-1]);
   if s[sp].itype = expr then
      disposenodes(s[sp].first,sp);
   decstack;
end (* addr,val  expr,val *)
else if (st1.itype=val)
   and ((st2.itype=addr) or (st2.itype=expr)) then
begin
   if st2.valu=1 then swapitem(s[sp],s[sp-1]);
      if s[sp].itype = expr then
         disposenodes(s[sp].first,sp);
      decstack;
   end (* val,addr  val,expr *)
end (* iorproc *);

procedure notproc;   (* perform boolean not *)
begin
   case s[sp].itype of
   val : s[sp].valu:=abs(s[sp].valu-1);
   addr,expr : s[sp].neg:=not s[sp].neg;
   end (* case *);
end (* notproc *);

(* - - other - - - - - - - - - - - - - - - - - - - - - *)

procedure mstproc;   (* creates mark stack information *)
begin
   incstack;
   with s[sp] do
   begin
      itype:=mst;
      dynlink:=mp;
      statlink:=searchaddr(arg1,0);
      return:=-1;
      neg:=false;
   end (* with *);
   semaphor:=semaphor+1;
end (* mstproc *);

procedure cupproc;   (* call user & external procedure *)
var   raddr,v,i:   integer;
      pbssp : pbsrange;
begin
   raddr:=pccount+1;   (* return address *)
   mp:=sp-arg1;
   case l[arg2].ltype of
   cup:
   begin
      for i:=mp+1 to sp do
      case s[i].itype of
      val:   ;
      addr:
         begin
            if s[i].neg then genpbs(2,s[i].address) (*adn*)
                    else genpbs(1,s[i].address);(*ad*)
            s[i].address:=getaddress(usedbefore);
            genpbs(5,s[i].address);   (* se *)
         end (*addr*);
      expr:
         begin
            pbssp:=getaddress(usedbefore);
            outnode(s[i].first);
            if s[i].neg then genpbs(6,pbssp)   (*sen*)
                    else genpbs(5,pbssp);   (*se*)
            disposenodes(s[i].first,i);
            s[i].itype:=addr;
            s[i].address:=pbssp;
```

```
      end (*expr*);
    end (* case *);
    pcskipto(l[arg2].level);
    compile([retpsym,retbsym,retisym]);
  end(* cup *);
  cxp:
    case l[arg2].name of
      timer:  timproc;
      origin: orgproc;
      edge:   exedgeproc(14);
      exor:   exedgeproc(15);
      pbsadof: paproc;
      clearpbs: clearproc;
    end(* case *);
  end(* case *);
  for i:=sp downto mp+1 do
  case s[i].itype of
    noitem,val,fil:   ;
    addr:    leaveaddress(s[i].address);
  end(* case *);
  (* Makes sp point to mst-element *)
  for i:=sp downto mp+1 do decstack;
  mp:=s[sp].dynlink;
  case instsym of
  retbsym:
        begin
          v:=s[sp].return;
          s[sp].itype:=addr;
          s[sp].address:=v;
          pushleave;
        end (* retb *);
  retisym:
        begin
          v:=s[sp].return;
          s[sp].itype:=val;
          s[sp].valu:=v;
        end(* reti *);
  retpsym:   decstack;
  end (* case *);
  semaphor:=semaphor-1;
  pcskipto(raddr);
end (* cupproc *);


procedure entproc; (* enter stack frame *)
var i:   integer;
begin
   if arg1=1 then
   begin
      if l[arg2].ltype()store then error(elab);
      sp:=mp+l[arg2].level-1;(*mst is included in ent-size*)
      chk(esp);
   end;
end (* entproc *);


procedure fjpproc;(* perform jump in pcfile if stacktop=0 *)
begin
   if s[sp].valu=0 then pcskipto(l[arg1].level);
   decstack;
end (* fjpproc *);


procedure ujpproc;(* perform jump in pcfile unconditional *)
begin
   pcskipto(l[arg1].level);
end (* ujpproc *);


procedure xjpproc; (* perform indexed jump in pcfile *)
begin
   pcskipto(l[arg1].level+s[sp].valu);
   decstack;
end(* xjpproc *);

procedure movproc; (* move data block *)
var   i,pto : integer;
begin
   for i:=0 to arg1-1 do
   begin
      pto:=s[sp-1].valu+i;
      with s[s[sp].valu+i] do
      case itype of
      noitem:    s[pto].itype:=noitem;
      val:
         begin
            s[pto].itype:=val;
            s[pto].valu:=valu;
         end;
      addr:
         begin
            assignaddress(pto,free);
            genpbs(1,address);
            genpbs(5,s[pto].address);
         end;
      end(* case *);
   end(* for *);
   decstack;
   decstack;
end (* movproc *);


procedure endproc;
begin
   genpbs(pbsnumbinstr,0); (* end *)
   writeln;
   if not errflag then write('no ');
   writeln('errors were found'); writeln;
   writeln('end of program.');
end (* endproc *);


procedure ifproc;
   procedure outcond;
   var    i:integer;
   begin
      if not (nextinstsym in [ifsym,elsesym,eifsym]) then
      begin
         for i:=1 to iflev do
```

```
         case ifarr[i] of
         thenstat :
           genpbs(1,octal(decimal(pbsifmin)+i-1));(*ad*)
         elsestat :
           genpbs(2,octal(decimal(pbsifmin)+i-1));(*adn*)
         end (* case *);
      if iflev>0 then genpbs(8,0); (*ssn*)
     end;
   end (* outcond *);

begin
   if iflev>0 then genpbs(9,0); (*rs*)
   compile([thensym]);
   case s[sp].itype of
   val:
   begin
      if s[sp].valu=0 then
      begin
         skip([elsesym,eifsym]);
         if instsym=elsesym then
         begin
            outcond;
            compile([eifsym]);
         end;
      end
      else
      begin
         outcond;
         compile([elsesym,eifsym]);
         if instsym=elsesym then skip([eifsym]);
      end;
   end (* val *);
   addr,expr:
   begin
      iflev:=iflev+1;
      chk(eif);
      ifarr[iflev]:=thenstat;
      if s[sp].itype=addr then
      begin
         if s[sp].neg then
            genpbs(2,s[sp].address) (*adn*)
         else
            genpbs(1,s[sp].address); (*ad*)
         genpbs(5,octal(decimal(pbsifmin)+iflev-1)); (*se*)
      end
      else
      begin
         outnode(s[sp].first);
         if s[sp].neg then
           genpbs(6,octal(decimal(pbsifmin)+iflev-1))(*sen*)
         else
           genpbs(5,octal(decimal(pbsifmin)+iflev-1));(*se*)
         disposenodes(s[sp].first,sp);
      end;
      outcond;
      compile([elsesym,eifsym]);
      if instsym=elsesym then
      begin
         genpbs(9,0); (*rs*)
         ifarr[iflev]:=elsestat;
         outcond;
           compile([eifsym]);
         end;
         iflev:=iflev-1;
      end (* addr,expr *);
      end (* case *);
      genpbs(9,0); (*rs*)
      outcond;
      decstack;
   end (* ifproc *);



(* - - compile - - - - - - - - - - - - - - - - - - - - *)


procedure compile;   (* (symset: tset) *)
begin
   insymbol;
   while not (instsym in symset ) do
   begin
      case instsym of
         ldobsym:   ldobproc;
         ldcbsym:   ldcproc;
         srobsym:   srobproc;
         andsym:    andproc;
         iorsym:    iorproc;
         notsym:    notproc;
         ifsym:     ifproc;
         laosym:    ldcproc;
         ldcisym:   ldcproc;
         chkisym:   chk(echki);
         stobsym:   stobproc;
         decisym:   deciproc;
         incisym:   incproc;
         ixasym:    ixaproc;
         ldoisym:   ldoproc;
         sroisym:   sroiproc;
         adisym:    adiproc;
         dvisym:    dviproc;
         mpisym:    mpiproc;
         sbisym:    sbiproc;
         modsym:    modproc;
         equisym:   equiproc;
         grtisym:   grtiproc;
         geqisym:   geqiproc;
         lesisym:   lesiproc;
         leqisym:   leqiproc;
         neqisym:   neqiproc;
         cupsym:    cupproc;
         indbsym:   indbproc;
         mstsym:    mstproc;
         strbsym:   strbproc;
         lodasym:   lodproc;
         entsym:    entproc;
         lodbsym:   lodbproc;
         ldasym:    ldaproc;
         incasym:   incproc;
         strasym:   strproc;
         strisym:   strproc;
```

```
          fjpsym:     fjpproc;
          ujpsym:     ujpproc;
          movsym:     movproc;
          lodisym:    lodproc;
          stpsym:     ;
          ujcsym:     error(ecase);
          xjpsym:     xjpproc;
          ngisym:     ngiproc;
          stoisym:    stoiproc;
          indisym:    indiproc;
          cxpsym:     cupproc;
          txtsym:     ldcproc;
          cspsym:     cspproc;
          ldoasym:    ldoproc;
          ldccsym:    ldcproc;
        end (* case *);
        insymbol;
    end (* while *);
end (* compile *);


procedure skip; (* (symset; tset) *)
begin
    repeat insymbol until instsym in symset;
end (* skip *);



(* - - mainprogram - - - - - - - - - - - - - - - - - - - *)


begin
   writeln;
   writeln('Code generator for PBS-mini - ver. june 81');
   initsimple;
   initarray;
   init2array;
   fileinit;
   pcscanner;
   compile([endsym]);
   endproc;
end.
```

APPENDIX 3

Listning av optimeringspasset

```
program pbsopt(pbsin,pbsout,input,output);


(**********************************************************
*                                                         *
*    Code optimization for SATT Electronics PBS-mini      *
*                                                         *
*    Runable on VAX 11/780 VMS version 2.3                *
*                                                         *
*    Authors: Sten Minor                                  *
*             Oskar Permvall                              *
*                                                         *
*    Date:    June 15, 1981                               *
*                                                         *
*    Dept of Automatic Control                            *
*    Lund Institute of Technology                         *
*                                                         *
**********************************************************)


const lsize=4;
      comlength=76;

type  instrsymbol = (nosym,adsym,adnsym,orsym,ornsym,
                     sesym,sensym,sssym,ssnsym,rssym,
                     ssrsym,rsrsym,cntpsym,cntmsym,
                     cotpsym,cotmsym,edgesym,exsym,
                     exnsym,apsym,rpsym,cpsym,comsym,
                     endsym);
      instrec = record
         instr : instrsymbol;
         arg : integer
      end;

      psymnode = ↑symnode;
      symnode = record
         next : psymnode;
         sym : instrsymbol;
         argum : integer
      end;

      txt = packed array[1..comlength] of char;
      ptextnode = ↑textnode;
      textnode = record
         next : ptextnode;
         com : txt
      end;

var   pbsin,pbsout : text;
      instructions : array [adsym..endsym] of
                     packed array [1..4] of char;

      arguments : array [adsym..endsym] of boolean;
      currinstr : packed array [1..4] of char;
      currsym : instrsymbol;
      currarg : integer;
      currtext : txt;
      look : array [1..lsize] of instrec;
      i : integer;
      ii : instrsymbol;

      symshifted,skipset : boolean;
      endreached,endtouched : boolean;

      textlist : record
         first,ltext : ptextnode
      end;

      symlist : record
         first,linstr : psymnode
      end;
      remif : psymnode;


procedure initvariables;
begin
   symshifted:=false;
   skipset:=false;
   endreached:= false;
   endtouched:=false;
   remif:=nil;
   with symlist do
   begin
      first:=nil;
      linstr:=nil;
   end(* with *);
   with textlist do
   begin
      first:=nil;
      ltext:=nil;
   end(* with *);
   for i:=1 to lsize do look[i].instr:= nosym;
   instructions[adsym]:=  'AD  ';    arguments[adsym]:=  true;
   instructions[adnsym]:= 'ADN ';    arguments[adnsym]:= true;
   instructions[orsym]:=  'OR  ';    arguments[orsym]:=  true;
   instructions[ornsym]:= 'ORN ';    arguments[ornsym]:= true;
   instructions[sesym]:=  'SE  ';    arguments[sesym]:=  true;
   instructions[sensym]:= 'SEN ';    arguments[sensym]:= true;
   instructions[sssym]:=  'SS  ';    arguments[sssym]:=  false;
   instructions[ssnsym]:= 'SSN ';    arguments[ssnsym]:= false;
   instructions[rssym]:=  'RS  ';    arguments[rssym]:=  false;
   instructions[ssrsym]:= 'SSR ';    arguments[ssrsym]:= true;
   instructions[rsrsym]:= 'RSR ';    arguments[rsrsym]:= true;
   instructions[cntpsym]:='CNT+';    arguments[cntpsym]:=true;
   instructions[cntmsym]:='CNT-';    arguments[cntmsym]:=true;
   instructions[cotpsym]:='COT+';    arguments[cotpsym]:=true;
   instructions[cotmsym]:='COT-';    arguments[cotmsym]:=true;
   instructions[edgesym]:='EDGE';    arguments[edgesym]:=true;
   instructions[exsym]:=  'EX  ';    arguments[exsym]:=  true;
   instructions[exnsym]:= 'EXN ';    arguments[exnsym]:= true;
   instructions[apsym]:=  'AP  ';    arguments[apsym]:=  false;
   instructions[rpsym]:=  'RP  ';    arguments[rpsym]:=  false;
   instructions[cpsym]:=  'CP  ';    arguments[cpsym]:=  false;
   instructions[comsym]:= '*C* ';    arguments[comsym]:= false;
   instructions[endsym]:= 'END ';    arguments[endsym]:= false;
end;
```

```
procedure outsym;
begin
   with symlist.first↑ do
   begin
      if sym in [sesym,sensym,sssym,ssnsym,rssym,
           ssrsym,rsrsym,cotpsym,cotmsym,cpsym]
      then write(pbsout,'*')
      else
         case sym of
            apsym   : write(pbsout,'(');
            rpsym   : write(pbsout,')');
            comsym : ;
            otherwise write(pbsout,' ');
         end;
      write(pbsout,instructions[sym]);
      if arguments[sym] then write(pbsout,argum:10);
      if sym=comsym then
      with textlist do
      begin
         write(pbsout,first↑.com);
         first:=first↑.next;
      end(* with *);
      writeln(pbsout);
   end(* with *);
end(* outsym *);


function readch:char;
var   ch : char;
begin
   read(pbsin,ch); readch:=ch;
end(* readch *);


procedure fileinit;

type  namestring = packed array [1..13] of char;
      filetype = (inp,outp);

var   inputname,outputname : namestring;
      status : integer;
      test : boolean;
      ch : char;

   procedure filename(n : packed array [integer] of char;
                      var st : integer); extern;

   procedure filecontrol(var name : namestring;
            var test : boolean; ftype : filetype);

   var   status,nrofchars,maxnrofchars : integer;
         dotincluded : boolean;

   begin
      name:='             ';
      maxnrofchars:=9;
      nrofchars:=1;
      dotincluded:=false;
      while (nrofchars<=maxnrofchars) and not eoln do
      begin
         read(ch);
         name[nrofchars]:=ch;
         if ch='.' then
         begin
            dotincluded:=true;
            maxnrofchars:=13;
         end;
         nrofchars:=nrofchars+1;
      end;
      readln;
      if ftype=inp then
      begin
         if not dotincluded then
         begin
            name[nrofchars]:='.';
            name[nrofchars+1]:='p';
            name[nrofchars+2]:='b';
            name[nrofchars+3]:='s';
         end;
         filename(name,status);
         test:=status)=0;
      end
      else  if not dotincluded then
            begin
               name[nrofchars]:='.';
               name[nrofchars+1]:='o';
               name[nrofchars+2]:='p';
               name[nrofchars+3]:='t';
            end;
   end;

begin
   repeat
      writeln('Inputfile ?');
      filecontrol(inputname,test,inp);
      if not test then writeln('Error - file does ',
                                'not exist: ',inputname);
   until test;
   writeln('Outputfile ?');
   filecontrol(outputname,test,outp);
   writeln('Inputfile: ',inputname);
   writeln('Outputfile: ',outputname);
   filename(inputname,status);
   open(pbsin,'$$$fil',old);
   reset(pbsin);
   filename(outputname,status);
   open(pbsout,'$$$fil',new);
   rewrite(pbsout);
end(* fileinit *);


procedure insym;
begin
   if not endtouched then
   begin
      for i:=1 to 4 do currinstr[i]:=readch;
      ii:=adsym;
      while instructions[ii]<>currinstr do ii:=succ(ii);
      currsym:= ii;
      if arguments[ii] then read(pbsin,currarg);
      if currsym=comsym then
```

```
      with textlist do
      begin
         if first=nil then
         begin
            new(first);
            ltext:=first;
            ltext↑.next:=first;
         end;
         if ltext↑.next=first then
         begin
            new(ltext↑.next);
            ltext↑.next↑.next:=first;
         end;
         with ltext↑ do
         begin
            for i:=1 to comlength do com[i]:=' ';
            i:=0;
            while not eoln(pbsin) and (i<comlength) do
            begin
               i:=i+1;
               com[i]:=readch;
            end(* while *);
         end(* with *);
         ltext:=ltext↑.next;
      end(* with *);
      readln(pbsin);
      endtouched:= currsym=endsym;
   end;
end(* insym *);


procedure shift;
begin
   if look[lsize].instr <> nosym then
   with symlist do
   begin
      symshifted:=true;
      if first=nil then
      begin
         new(first);
         linstr:=first;
         linstr↑.next:=first;
      end
      else
         if linstr↑.next=first then
         begin
            new(linstr↑.next);
            linstr↑.next↑.next:=first;
            linstr:=linstr↑.next;
         end
         else linstr:=linstr↑.next;
      with linstr↑,look[lsize] do
      begin
         sym:=instr;
         if arguments[instr] then argum:=arg;
      end(* with *);
      endreached:=linstr↑.sym=endsym;
   end(* with *)
   else symshifted:=false;
   for i:=lsize-1 downto 1 do look[i+1]:= look[i];
   look[1].instr:= currsym;
   look[1].arg:= currarg;
end(* shift *);

procedure optimize;
begin

   if (look[3].instr=sensym) and
      (look[2].arg=look[3].arg) and
      (look[2].instr=adsym) and
      (look[1].instr=exsym) then
   begin
      look[2].instr:=nosym;
      look[3].instr:=nosym;
      look[1].instr:=exnsym;
   end;

   if (look[3].instr in [edgesym,exsym,exnsym]) and
      (look[2].instr=sesym) and
      (look[1].instr = adsym) and
      (look[2].arg=look[1].arg) then
   begin
      look[2].instr:=nosym;
      look[1].instr:=nosym;
   end;

   if (look[4].instr=edgesym) and
      (look[3].instr=sesym) and
      (look[2].instr=adnsym) and
      (look[1].instr=sesym) and
      (look[2].arg=look[3].arg) then
   begin
      look[2].instr:=nosym;
      look[3].instr:=nosym;
      look[1].instr:=sensym;
   end;

   if (look[3].instr in [exsym,exnsym]) and
      (look[2].instr=sesym) and
      (look[1].instr=adnsym) and
      (look[1].arg=look[2].arg) then
   begin
      case look[3].instr of
      exnsym:  look[3].instr:=exsym;
      exsym:   look[3].instr:=exnsym;
      end(* case *);
      look[2].instr:=nosym;
      look[1].instr:=nosym;
   end;

   if (look[1].instr=rssym) and
      ((look[2].instr=sesym) or (look[2].instr=sensym)) and
      (look[3].instr=ssnsym) then
   begin
      skipset:=false;
      with look[2] do
      case instr of
      sesym:   instr:=ssrsym;
      sensym:   instr:=rsrsym;
      end(* case *);
      look[1].instr:=nosym;
```

```
      look[3].instr:=nosym;
   end;

   with look[lsize] do
   begin
      if (instr=sssym) or (instr=ssnsym) then skipset:=true
      else if (instr=rssym) and skipset then skipset:=false
      else if (instr=rssym) and not skipset
         then instr:= nosym;
   end(* with *);
end(* optimize *);


procedure optim2;
begin
   if (symlist.first<>nil) and symshifted then
   with symlist do
   begin
      if (linstr↑.sym in [sesym,sensym,adsym,adnsym]) and
         ((linstr↑.argum=1004) or (linstr↑.argum=1005))
      then
         if remif<>nil
         then
            if linstr↑.argum=1004
            then
               if linstr↑.sym in [sesym,sensym]
               then
                  if remif↑.sym=sesym then
                  begin
                     first:=remif↑.next↑.next;
                     remif:=linstr;
                  end
                  else
                     if remif↑.next↑.next↑.sym=ssnsym then
                     begin
                        first:=remif↑.next↑.next;
                        first↑.sym:=sssym;
                        remif:=linstr;
                     end
                     else remif:=linstr
               else begin
                     if remif↑.next<>linstr
                     then remif:=nil
                  end
            else remif:=nil
      else if (linstr↑.argum=1004) and
              (linstr↑.sym in [sesym,sensym])
           then remif:=linstr;

      if remif<>nil
      then
         while first<>remif do
         begin
            outsym;
            first:=first↑.next;
         end
      else
         while first<>linstr do
         begin
            outsym;
            first:=first↑.next;
         end;

      if endreached then
      begin
         if remif<>nil
         then
            if remif↑.sym=sesym
            then first:=remif↑.next↑.next
            else if remif↑.next↑.next↑.sym=ssnsym then
                    begin
                       first:=remif↑.next↑.next;
                       first↑.sym:=sssym;
                    end;
         while linstr<>first do
         begin
            outsym;
            first:=first↑.next;
         end(* while *);
         outsym;
      end;
   end(* with *);
end(* optim2 *);

begin  (* main *)
   writeln;
   writeln('Pbsopt - ver.',' june 81');
   initvariables;
   fileinit;
   while not endreached do
   begin
      insym;
      shift;
      optimize;
      optim2;
   end;
   writeln;writeln('end of program.');
   close(pbsin);
   close(pbsout);
end.
```