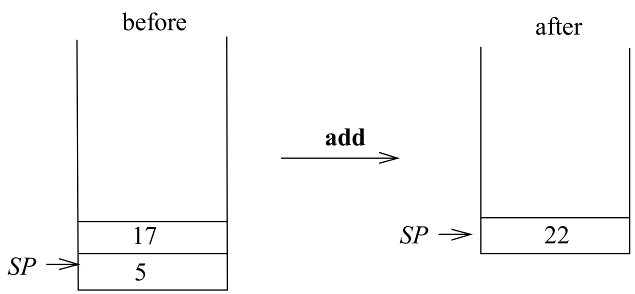
P-instructions for Arithmetic

Instr.	Meaning	Cond.	Res.
add N	$STORE[SP-1] := STORE[SP-1] +_N STORE[SP]$; $SP := SP - 1$	$\begin{pmatrix}N\\N\end{pmatrix}$	(N)
sub N	$STORE[SP-1] := STORE[SP-1] -_N STORE[SP]$; $SP := SP - 1$	$\begin{pmatrix}N\\N\end{pmatrix}$	(N)
mul N	$STORE[SP-1] := STORE[SP-1] *_N STORE[SP]$; $SP := SP - 1$	$\begin{pmatrix}N\\N\end{pmatrix}$	(N)
div N	$STORE[SP-1] := STORE[SP-1] /_N STORE[SP]$; $SP := SP - 1$	$\begin{pmatrix}N\\N\end{pmatrix}$	(N)
neg N	$STORE[SP] := -_N STORE[SP]$	(N)	(N)



P-instructions for Boolean Operations

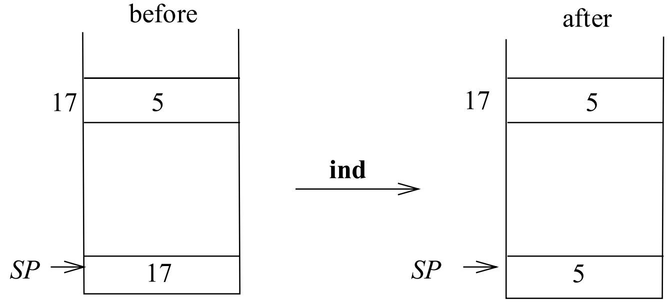
Instr.	Meaning	Cond.	Res.
and	$STORE[SP-1] := STORE[SP-1]$ *and* $STORE[SP]$; $SP := SP - 1$	$\begin{pmatrix}b\\b\end{pmatrix}$	(b)
or	$STORE[SP-1] := STORE[SP-1]$ *or* $STORE[SP]$; $SP := SP - 1$	$\begin{pmatrix}b\\b\end{pmatrix}$	(b)
not	$STORE[SP] :=$ *not* $STORE[SP]$	(b)	(b)

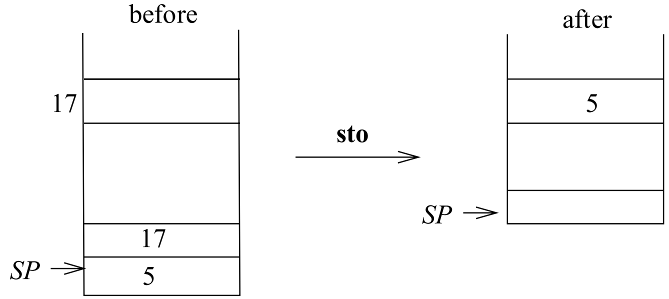
P-instructions for comparisons

Instr.	Meaning	Cond.	Res.
equ T	$STORE[SP-1] := STORE[SP-1] =_T STORE[SP]$; $SP := SP-1$	$\begin{pmatrix}T\\T\end{pmatrix}$	(b)
geq T	$STORE[SP-1] := STORE[SP-1] \geq_T STORE[SP]$; $SP := SP-1$	$\begin{pmatrix}T\\T\end{pmatrix}$	(b)
leq T	$STORE[SP-1] := STORE[SP-1] \leq_T STORE[SP]$; $SP := SP-1$	$\begin{pmatrix}T\\T\end{pmatrix}$	(b)
les T	$STORE[SP-1] := STORE[SP-1] <_T STORE[SP]$; $SP := SP-1$	$\begin{pmatrix}T\\T\end{pmatrix}$	(b)
grt T	$STORE[SP-1] := STORE[SP-1] >_T STORE[SP]$; $SP := SP-1$	$\begin{pmatrix}T\\T\end{pmatrix}$	(b)
neq T	$STORE[SP-1] := STORE[SP-1] \neq_T STORE[SP]$; $SP := SP-1$	$\begin{pmatrix}T\\T\end{pmatrix}$	(b)

P-instructions for load/store

Instr.	Meaning	Cond.	Res.
ldo $T\,q$	$SP := SP+1$; $STORE[SP] := STORE[q]$	$q \in [0, maxstr]$	(T)
ldc $T\,q$	$SP := SP+1$; $STORE[SP] := q$	$Typ(q) = T$	(T)
ind T	$STORE[SP] := STORE[STORE[SP]]$	(a)	(T)
sro $T\,q$	$STORE[q] := STORE[SP]$; $SP := SP-1$	(T) $q \in [0, maxstr]$	
sto T	$STORE[STORE[SP-1]] := STORE[SP]$; $SP := SP-2$	$\begin{pmatrix}a\\T\end{pmatrix}$	





The translation of assignments and expressions

Function		Condition
$code_R(e_1 = e_2)\ \rho$	$= code_R\ e_1\ \rho; code_R\ e_2\ \rho;$ **equ** T	$Typ(e_1) = Typ(e_2) = T$
$code_R(e_1 \neq e_2)\ \rho$	$= code_R\ e_1\ \rho; code_R\ e_2\ \rho;$ **neq** T	$Typ(e_1) = Typ(e_2) = T$
$\vdots$		
$code_R(e_1 + e_2)\ \rho$	$= code_R\ e_1\ \rho; code_R\ e_2\ \rho;$ **add** N	$Typ(e_1) = Typ(e_2) = N$
$code_R(e_1 - e_2)\ \rho$	$= code_R\ e_1\ \rho; code_R\ e_2\ \rho;$ **sub** N	$Typ(e_1) = Typ(e_2) = N$
$code_R(e_1 * e_2)\ \rho$	$= code_R\ e_1\ \rho; code_R\ e_2\ \rho;$ **mul** N	$Typ(e_1) = Typ(e_2) = N$
$code_R(e_1 / e_2)\ \rho$	$= code_R\ e_1\ \rho; code_R\ e_2\ \rho;$ **div** N	$Typ(e_1) = Typ(e_2) = N$
$code_R(-e)\ \rho$	$= code_R\ e\ \rho;$ **neg** N	$Typ(e) = N$
$code_R\ x\ \rho$	$= code_L\ x\ \rho;$ **ind** T	x is a variable of type T
$code_R\ c\ \rho$	$=$ **ldc** T c	c is a constant of type T
$code(x := e)\ \rho$	$= code_L\ x\ \rho;\ code_R\ e\ \rho;$ **sto** T	$Typ(e) = T$,
		x is a variable
$code_L\ x\ \rho$	$=$ **ldc** a $\rho(x)$	x is a variable

Conditional Statements

Machine ressources to implement control statements: unconditional and conditional jumps

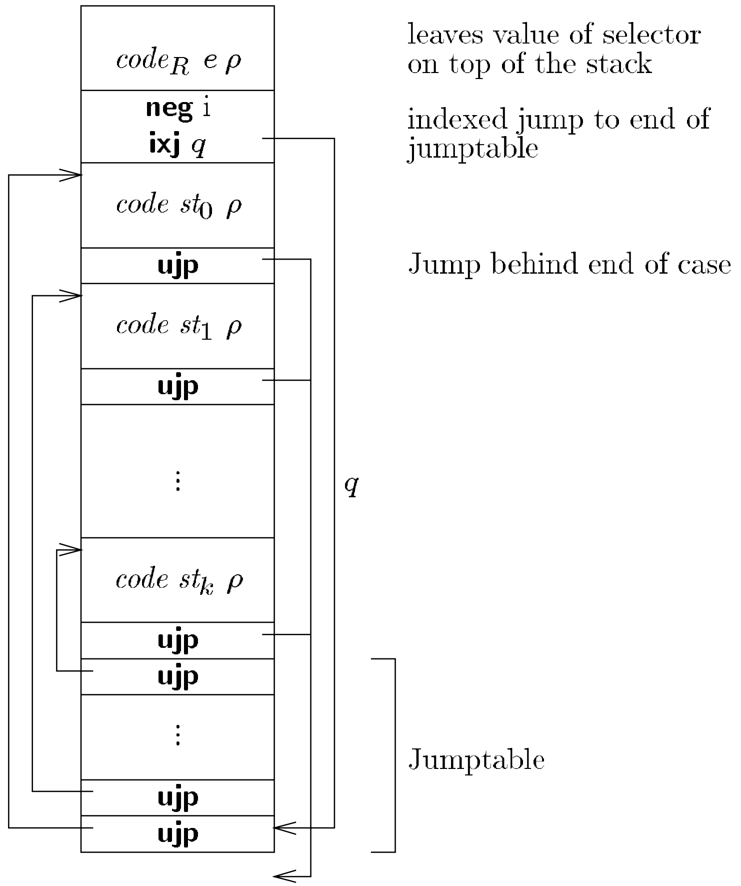
P-code Instructions for Branches

Inst.	Meaning	Cond.	Result
ujp q	$PC := q$	$q \in [0, codemax]$	
fjp q	**if** $STORE[SP] = false$ **then** $PC := q$ **fi**; $SP := SP - 1$	(b) $q \in [0, codemax]$	$()$

fjp consumes the boolean value on top of the stack.

P-Code Instructions for Jump Table

Instr.	Meaning	Cond.	Result
ixj q	$PC := STORE[SP] + q;$ $SP := SP - 1$	(i)	$()$



P-Code for Array Indexing

Instr.	Meaning	Cond.	Result
ixa q	$STORE[SP-1] :=$ $STORE[SP-1] +$ $STORE[SP] * q;$ $SP := SP - 1$	$\binom{i}{a}$	(a)
inc $T q$	$STORE[SP] :=$ $STORE[SP] + q$	$(T),\ Typ(q) = i$	(T)
dec $T q$	$STORE[SP] :=$ $STORE[SP] - q$	$(T),\ Typ(q) = i$	(T)

$$
\begin{aligned}
code_L\ b[i_1, \ldots, i_n]\ g\ \rho &= \textbf{ldc}\ a\ \rho(b);\ code_I\ [i_1, \ldots, i_n]\ g\ \rho \\
code_I\ [i_1, \ldots, i_n]\ g\ \rho &= code_R\ i_1\ \rho;\ \textbf{ixa}\ g * d^{(1)}; \\
&\quad code_R\ i_2\ \rho;\ \textbf{ixa}\ g * d^{(2)}; \\
&\quad \vdots \\
&\quad code_R\ i_n\ \rho;\ \textbf{ixa}\ g; \\
&\quad \textbf{dec}\ a\ g * d;
\end{aligned}
$$

P-code for Array Checking

Instr.	Meaning	Cond.	Result
chk $p\ q$	**if not** $(p \leq STORE[SP] \leq q)$ **then** *error*(*"value out of range"*) **fi**	(i)	(i)

New code for indexing including array bound checks

$code_I\ [i_1, \ldots, i_n]\ desc\ \rho =$
$code_R\ i_1\ \rho;$ **chk** $l_1\ u_1;$ **ixa** $g \cdot d^{(1)};$
$code_R\ i_2\ \rho;$ **chk** $l_2\ u_2;$ **ixa** $g \cdot d^{(2)};$
$\vdots$
$code_R\ i_n\ \rho;$ **chk** $l_n\ u_n;$ **ixa** $g;$
dec a $g \cdot d;$

where $desc = (g; l_1, u_1, \ldots, l_n, u_n)$

This array description is made available through the symbol table, c.f. Semantic Analysis.

Array Descriptor

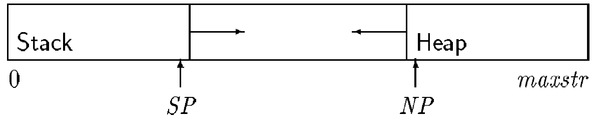
0	Adjusted Address :a
1	Array size :i
2	Subtr. part :i
3	l_1 :i
4	u_1 :i
$\vdots$	$\vdots$
$2n+1$	l_n :i
$2n+2$	u_n :i
$2n+3$	d_2 :i
$\vdots$	$\vdots$
$3n+1$	d_n :i

$code_{Ld}\ c[i_1, \ldots, i_k]\ \rho =$ **ldc** a $\rho(c);$ $code_{Id}\ [i_1, \ldots, i_k]\ g\ \rho$
$code_{Id}\ [i_1, \ldots, i_n]\ g\ \rho\ =$
dpl i; descriptor address
ind i; adjusted address
ldc i $0;$
$code_R\ i_1\ \rho;$ **add** i; **ldd** $2n+3;$ **mul** i;
$code_R\ i_2\ \rho;$ **add** i; **ldd** $2n+4;$ **mul** i;
$\ldots$
$code_R\ i_{n-1}\ \rho;$ **add** i; **ldd** $3n+1;$ **mul** i;
$code_R\ i_n\ \rho;$ **add** i;
ixa $g;$
sli a

Additional P-code instructions

Instr.	Meaning	Cond.	Result
dpl T	$SP := SP + 1;$ $STORE[SP] := STORE[SP-1]$	T	$\begin{pmatrix} T \\ T \end{pmatrix}$
ldd q	$SP := SP + 1;$ $STORE[SP] := STORE[STORE[SP-3]+q]$	$\begin{pmatrix} T_2 \\ T_1 \\ a \end{pmatrix}$	$\begin{pmatrix} i \\ T_2 \\ T_1 \\ a \end{pmatrix}$
sli T_2	$STORE[SP-1] := STORE[SP];$ $SP := SP - 1$	$\begin{pmatrix} T_2 \\ T_1 \end{pmatrix}$	(T_1)

The Heap of the P-Machine



Instruct.	Meaning	Cond.	Result
new	**if** $NP - STORE[SP] \leq EP$ **then** *error* (“store overflow”) **else** $NP := NP - STORE[SP];$ $STORE[STORE[SP-1]] := NP;$ $SP := SP - 2$ **fi**;	$\begin{pmatrix} i \\ a \end{pmatrix}$	

Procedure Call

$code\ p(e_1, \ldots, e_k)\ \rho\ nd =$
  mst $nd - nd'$;
  $code_A\ e_1\ \rho\ nd$;
  ⋮
  $code_A\ e_k\ \rho\ nd$;
  cup $s\ l$ (* $\rho(p) = (l, nd)$ *)

P-instructions for call and entry

Instr.	Meaning	Comment
mst p	$STORE[SP+2] := base(p, MP)$; $STORE[SP+3] := MP$; $STORE[SP+4] := EP$; $SP := SP + 5$	Static link Dynamic link Save EP
cup $p\ q$	$MP := SP - (p+4)$; $STORE[MP+4] := PC$; $PC := q$	p space for parameters Return address Jump to q
ssp p	$SP := MP + p - 1$	alloc. static area
sep p	$EP := SP + p$; **if** $EP \geq NP$ **then** *error* (*"store overflow"*) **fi**	alloc. temp. area Check collision stack and heap

$base(p, a)$ = **if** $p = 0$ **then** a **else** $base(p-1, STORE[a+1])$**fi**

Adapt $code_L$

Instr.	Meaning
lod $T\ p\ q$	$SP := SP + 1$; $STORE[SP] := STORE[base(p, MP) + q]$
lda $p\ q$	$SP := SP + 1$; $STORE[SP] := base(p, MP) + q$
str $T\ p\ q$	$STORE[base(p, MP) + q] := STORE[SP]$; $SP := SP - 1$

$base(p, a)$ = **if** $p = 0$ **then** a **else** $base(p-1, STORE[a+1])$

$code_L(x\ r)\ \rho\ nd =$ **lda** a $d\ ra$;
  $code_M\ r\ \rho\ nd$,
  where $\rho(x) = (ra, nd')$, $d = nd - nd'$ and
  x is variable or formal value parameter

$code_L(x\ r)\ \rho\ nd =$ **lod** a $d\ ra$;
  $code_M\ r\ \rho\ nd$
  where $\rho(x) = (ra, nd')$, $d = nd - nd'$,
  and x is formal var parameter

Code for Procedure/Function Return

1. Restore SP to the beginning of current stack frame
2. Restore PC to return-address
3. Restore EP and check for heap-stack collision
4. Release frame, i.e. set MP to dynamic link

P-instructions for Return

Instr.	Meaning	Comment
retf	$SP := MP$; $PC := STORE[MP + 4]$; $EP := STORE[MP + 3]$; **if** $EP \geq NP$ **then** *error*(*"store overflow"*) **fi** $MP := STORE[MP + 2]$	result is on top return address Restore EP Release frame
retp	$SP := MP - 1$; $PC := STORE[MP + 4]$; $EP := STORE[MP + 3]$; **if** $EP \geq NP$ **then** *error*(*"store overflow"*) **fi** $MP := STORE[MP + 2]$	No return value return address Restore EP Release frame

P-code for moves

Instr.	Meaning	Cond.	Res.
movs q	**for** $i := q - 1$ **down to** 0 **do** $STORE[SP + i] :=$ $STORE[STORE[SP] + i]$ **od**; $SP := SP + q - 1$	(a)	
movd q	**for** $i := 1$ **to** $STORE[MP + q + 1]$ **do** $STORE[SP + i] :=$ $STORE[STORE[MP + q]$ $+ STORE[MP + q + 2] + i - 1]$ **od**; $STORE[MP + q] :=$ $SP + 1 - STORE[MP + q + 2]$ $SP := SP + STORE[MP + q + 1]$		

Copying Dynamic Arrays

$code_P$ (**value** x : **array**$[u_1..o_1, \ldots, u_k..o_k]$ **of** t) ρ nd =
movd ra;