```
mask AuxMask

aluOp[5:0] dispatch[6:6] to-page-zero[7:7] mtyp[9:8] use_aluOp_h[10:10] alusourc
e[13:11]
setpriority[15:14] datsel[19:16];

endmask


mask MainMask

datinsel[27:24] addr_sel[23:20] next_pc[19:18] xinp2sel[17:16]
xinp1sel[15:14] dspalt[13:13] cspalt[12:12] mic_jmp_allow[11:11]
set_status[9:9]
readmem[7:7] writemem[6:6] next_mic[3:0];

endmask

values setpriority
default 0 no
1 yes
2 reenable
3 noop
endvalues




values use_aluOp_h

default 0 no
1 yes

endvalues


values aluOp

0 plus
1 minus
2 lte
3 gte
4 lt
5 gt
6 eq

16 tag-and
17 tag-ior
18 tag-xor

19 tag-eq
20 get-tag
21 tag-set

24 addr-eq
25 addr-eqi
26 addr-and
27 addr-ior
28 get-addr

32 palign3
```



```
33 calculateLenVect
34 setVectTag
35 plusAddr

36 compAddr

37 pConstr
38 pAdd

39 pShiftl
40 pShiftr

42 PutVal16
43 getTemplateAddr
44 getLitBaseAddr
45 getTpCodeStart
46 printAddr

50 getInfo


default 63 opB

endvalues



values alusource

default 0 regA
1 val12
2 val8
3 databus

endvalues


values mtyp

default 0 full
1 half
3 byte

endvalues



values addr_sel

default 0 regA
1 litidx
2 aluOut
3 tos
4 pc
5 litbase
6 tp
7 regB
8 csp
9 csp_m4
10 csp_d
11 csp_dm4
```

```
12 dsp
13 dsp_m4
14 dsp_d
15 dsp_dm4

endvalues

values next_pc

default 0 oldpc
1 accux
2 databus

endvalues

/*
values addr_en

default 0 no
1 yes
endvalues
*/

values datsel

default 0 dsp
1 csp
2 aluOut
3 datain
4 tos
5 d
6 tp
7 litbase
8 regB
9 regC
10 status
11 datain2
12 regB2
13 regA
14 pc
15 val8

endvalues

values datinsel

0 dsp
1 csp
2 regA
3 regB
4 regC
5 tos
6 tp
7 litbase
8 ir
default 9 none
10 regATo0
11 regATo4
12 tosTo4

endvalues
```



```
values xinp1sel

default 0 pc
1 dsp
2 csp

endvalues


values xinp2sel

default 0 val4
1 val_4
2 d

endvalues


values dspalt

default 0 no
1 yes

endvalues



values cspalt

default 0 no
1 yes

endvalues



values mic_jmp_allow

default 0 no
1 yes

endvalues



values readmem

default 0 no
1 yes

endvalues

values writemem

default 0 no
1 yes

endvalues

values dispatch

default 0 no
```

```
1 yes

endvalues

values to-page-zero

default 0 no
1 yes

endvalues


values set_status

default 0 no
1 yes

endvalues

/*

Der Anfang des Microcodes: Instruction fetch:

*/

:START:0:

addr_sel = pc, readmem = yes, next_mic = +;

datsel = datain, datinsel = ir, dispatch = yes, next_mic = 0;

/*




1. Kontrolltransfer
   ================

1.1. Sprungbefehle
     =============

JNIL d      Wenn <(DSP++)> = NIL then PC := PC + d else PC := PC + 4

*/

:JNIL:11:

mic_jmp_allow = yes, next_mic = +;
xinp1sel = pc, xinp2sel = val4, next_pc = accux, next_mic = !;
xinp1sel = pc, xinp2sel = d, next_pc = accux, next_mic = +;
addr_sel = dsp, readmem = yes, xinp1sel = dsp, xinp2sel = val4, dspalt = yes, ne
xt_mic = +;
datsel = datain, datinsel = tos, to-page-zero = yes, next_mic = 0;


/*

pcbsource = pc-jnil,
addro = dsp, readmem = yes,
cdspsource = dsp_4, cdspdest = dsp_set_cdspbus,
```



```
next_mic = +;

datsource = datamem, datdest = tos, to-page-zero = yes, next_mic = 0;
*/

/*


JMP d       PC := PC + d

*/


:JMP:12:

xinp1sel = pc, xinp2sel = d, next_pc = accux, to-page-zero = yes, next_mic = 0;

/*

pcbsource = pc+d, to-page-zero = yes, next_mic = 0;

*/


/*
Interrupt-Routinen-Aufruf
=========================

(note that due to a quirk in the microcode organization blocks with more than 8
muops have to
start at even numbers)

*/

:IRQCALLCLOS:14:

/* datsource = tos, datdest = regE, setpriority = yes, next_mic = +; */

use_aluOp_h = yes, aluOp = PutVal16, setpriority = yes, next_mic = +;

addr_sel = aluOut, readmem = yes, next_mic = +;


/* regA gets loaded with (16) = irq-closure */
datsel = datain, datinsel = regA, xinp1sel = dsp, xinp2sel = val_4, dspalt = yes
, next_mic = +;


/* put regA = (16), then tos on DS */

addr_sel = dsp, writemem = yes, datsel = regA, xinp1sel = dsp, xinp2sel = val_4,
 dspalt = yes, next_mic = +;

addr_sel = dsp, writemem = yes, datsel = tos, xinp1sel = csp, xinp2sel = val_4,
cspalt = yes, next_mic = +;


/* saving tp, then pc on CS */
addr_sel = csp, datsel = tp, writemem = yes, xinp1sel = csp, xinp2sel = val_4, c
spalt = yes, next_mic = +;

addr_sel = csp, datsel = pc, writemem = yes, next_mic = +;
```

```
/* write 8 to tos, get templateAddress to tp */
use_aluOp_h = yes, aluOp = getTemplateAddr, datsel = val8, datinsel = tos, next_
mic = +;
addr_sel = aluOut, readmem = yes, next_mic = +;
datsel = datain, datinsel = tp, next_mic = +;

/* write tp to regA */
datsel = tp, datinsel = regA, next_mic = +;

/* compute litbase */
use_aluOp_h = yes, aluOp = getLitbaseAddr, next_mic = +;
datsel = aluOut, datinsel = litbase, use_aluOp_h = yes, aluOp = getTpCodeStart,
next_mic = +;

/* compute code start */
addr_sel = aluOut, readmem = yes, next_mic = +;
datsel = datain, next_pc = databus, to-page-zero = yes, next_mic = 0;




/*


1.2. Closure-Calls
     =============

CALLCLOS d    (d = Anzahl der Parameter)


*/

:CALLCLOS:16:


/* write tos out to DSP */
datsel = tos, addr_sel = dsp_m4, writemem = yes, next_mic = +, xinp1sel = pc, xi
np2sel = val4, next_pc = accux;
/* set tos to d */
datsel = d, datinsel = tos, xinp1sel = csp, xinp2sel = val_4, cspalt = yes, next
_mic = +;
/*save tp */
datsel = tp, addr_sel = csp, writemem = yes, xinp1sel = csp, xinp2sel = val_4, c
spalt = yes, next_mic = +;
/* save pc + 4 */
datsel = pc, addr_sel = csp, xinp1sel = dsp, xinp2sel = val_4, dspalt = yes, wri
temem = yes, next_mic = +;

/* fetch closure regA := (dsp + d - 4) */
addr_sel = dsp_dm4, readmem = yes, next_mic = +;
datsel = datain, datinsel = regA, next_mic = +;

/* get templateAddress to tp */
use_aluOp_h = yes, aluOp = getTemplateAddr, next_mic = +;
addr_sel = aluOut, readmem = yes, next_mic = +;
datsel = datain, datinsel = tp, next_mic = +;

/* write tp to regA */
datsel = tp, datinsel = regA, next_mic = +;

/* compute litbase = tp + 16 */
```



```
use_aluOp_h = yes, aluOp = getLitbaseAddr, next_mic = +;
datsel = aluOut, datinsel = litbase, next_mic = +;

/* compute code start (tp + 4) */
use_aluOp_h = yes, aluOp = getTpCodeStart, next_mic = +;
addr_sel = aluOut, readmem = yes, next_mic = +;
datsel = datain, next_pc = databus, to-page-zero = yes, next_mic = 0;






/*


CALLCLOSV     (Anzahl Parameter liegt auf (DSP))

*/

:CALLCLOSV:18:


/* pc := pc + 4, csp := csp - 4 */
xinp1sel = pc, xinp2sel = val4, next_pc = accux, next_mic = +;
/*save tp */
datsel = tp, addr_sel = csp, writemem = yes, xinp1sel = csp, xinp2sel = val_4, c
spalt = yes, next_mic = +;
/* save pc + 4 */
datsel = pc, addr_sel = csp, writemem = yes, next_mic = +;

/* fetch closure regA := (dsp + d - 4) */
addr_sel = dsp_dm4, readmem = yes, next_mic = +;
datsel = datain, datinsel = regA, next_mic = +;

/* get templateAddress to tp */
use_aluOp_h = yes, aluOp = getTemplateAddr, next_mic = +;
addr_sel = aluOut, readmem = yes, next_mic = +;
datsel = datain, datinsel = tp, next_mic = +;

/* write tp to regA */
datsel = tp, datinsel = regA, next_mic = +;

/* compute litbase = tp + 16 */
use_aluOp_h = yes, aluOp = getLitbaseAddr, next_mic = +;
datsel = aluOut, datinsel = litbase, next_mic = +;

/* compute code start tp + 4 */
use_aluOp_h = yes, aluOp = getTpCodeStart, next_mic = +;
addr_sel = aluOut, readmem = yes, next_mic = +;
datsel = datain, next_pc = databus, to-page-zero = yes, next_mic = 0;



/*


1.3  Returns
     =======

RET
```

```
*/


:RET:20:

addr_sel = csp, readmem = yes, xinp1sel = csp, xinp2sel = val4, cspalt = yes,
setpriority = reenable, next_mic = +;

datsel = datain, next_pc = databus, addr_sel = csp, readmem = yes,
xinp1sel = csp, xinp2sel = val4, cspalt = yes, next_mic = +;

datsel = datain, datinsel = tp,  next_mic = +;


/* write tp to regA */
datsel = tp, datinsel = regA, next_mic = +;

/* compute litbase = tp + 16 */
use_aluOp_h = yes, aluOp = getLitBaseAddr, next_mic = +;
datsel = aluOut, datinsel = litbase, to-page-zero = yes, next_mic = 0;


/*



1.4. Halts und Interrupts
     ====================

HALT

*/


:HALT:30:

next_mic = 0;



/*


EXCEPT d


2. Datentransfers
   ==============

2.1. Innerhalb des Datenstacks
     =========================

LOD d      <DSP[d]> -> <--DSP>

*/

:LOD:1:


addr_sel = dsp_m4, datsel = tos, writemem = yes, xinp1sel = dsp, xinp2sel = val_
4, dspalt = yes, next_mic = +;
addr_sel = dsp_d, readmem = yes, next_mic = +;
```



```
datsel = datain, datinsel = tos, to-page-zero = yes, xinp1sel = pc, xinp2sel = v
al4, next_pc = accux, next_mic = 0;


/*


STO d      <DSP[0]> -> <(++DSP)[d-1]>

*/

:STO:3:


addr_sel = dsp_dm4, datsel = tos, writemem = yes, next_mic = +;
addr_sel = dsp, readmem = yes, xinp1sel = dsp, xinp2sel = val4, dspalt = yes, ne
xt_mic = +;
datsel = datain, datinsel = tos, to-page-zero = yes, xinp1sel = pc, xinp2sel = v
al4, next_pc = accux, next_mic = 0;

/*

LODNIL  NIL -> <(++DSP)>

*/



:LODNIL:2:


addr_sel = dsp_m4, datsel = tos, writemem = yes, xinp1sel = dsp, xinp2sel = val_
4, dspalt = yes, next_mic = +;
datsel = regC, datinsel = tos, to-page-zero = yes, xinp1sel = pc, xinp2sel = val
4, next_pc = accux, next_mic = 0;



/*



2.2. Zwischen Daten- und Kontrollstack
     =================================

LODCS d    <CSP[d]> -> <--DSP>

STOCS d    <DSP++> -> <CSP[d]>

PUSHCS d   <DSP[d]> -> <--CSP>

*/

:LODCS:7:


addr_sel = dsp_m4, datsel = tos, writemem = yes, xinp1sel = dsp, xinp2sel = val_
4, dspalt = yes, next_mic = +;
addr_sel = csp_d, readmem = yes, next_mic = +;
datsel = datain, datinsel = tos, to-page-zero = yes, xinp1sel = pc, xinp2sel = v
al4, next_pc = accux, next_mic = +;
```

```
:STOCS:8:

addr_sel = csp_d, datsel = tos, writemem = yes, next_mic = +;
addr_sel = dsp, readmem = yes, xinp1sel = dsp, xinp2sel = val4, dspalt = yes, ne
xt_mic = +;
datsel = datain, datinsel = tos, to-page-zero = yes, xinp1sel = pc, xinp2sel = v
al4, next_pc = accux, next_mic = +;



:PUSHCS:9:

addr_sel = dsp_m4, datsel = tos, writemem = yes, xinp1sel = dsp, xinp2sel = val_
4, dspalt = yes, next_mic = +;
addr_sel = dsp_d, readmem = yes, next_mic = +;
datsel = datain, datinsel = tos, next_mic = +;

addr_sel = csp_m4, writemem = yes, datsel = tos, xinp1sel = csp, xinp2sel = val_
4, cspalt = yes, next_mic = +;
xinp1sel = dsp, xinp2sel = val4, dspalt = yes, next_mic = +;
to-page-zero = yes, xinp1sel = pc, xinp2sel = val4, next_pc = accux, next_mic =
0;


/*

2.3. Vom aktuellen Template
     ======================

LITIDX d  <--DSP> := <LITBASE[d]>


*/

:LITIDX:4:


addr_sel = dsp_m4, datsel = tos, writemem = yes, xinp1sel = dsp, xinp2sel = val_
4, dspalt = yes, next_mic = +;
addr_sel = litidx, readmem = yes, next_mic = +;
datsel = datain, datinsel = tos, to-page-zero = yes, xinp1sel = pc, xinp2sel = v
al4, next_pc = accux, next_mic = 0;


/*


3. Änderungen der Stapelzeiger
   ===========================

INCSP d   DSP := DSP + d

DECSP d   DSP := DSP - d

INCCSP d  CSP := CSP + d

*/


:INCSP:5:
```



```
xinp1sel = dsp, xinp2sel = val_4, dspalt = yes, next_mic = +;

addr_sel = dsp, writemem = yes, datsel = tos, xinp1sel = dsp, xinp2sel = d, dspa
lt = yes,
next_mic = +;

addr_sel = dsp, readmem = yes, xinp1sel = dsp, xinp2sel = val4, dspalt = yes, ne
xt_mic = +;
datsel = datain, datinsel = tos, to-page-zero = yes,
xinp1sel = pc, xinp2sel = val4, next_pc = accux, next_mic = 0;


:DECSP:6:

/*

cdspsource = dsp_m_4, cdspdest = dsp_set_cdspbus, next_mic = +;

addro = dsp, datsource = tos, writemem = yes, next_mic = +;

cdspsource = dsp_minus_d, cdspdest = dsp_set_cdspbus, next_mic = +;

addro = dsp, readmem = yes, cdspsource = dsp_4, cdspdest = dsp_set_cdspbus, next
_mic = +;

datsource = datamem, datdest = tos, pcbsource = pc+4,
to-page-zero = yes, next_mic = 0;


*/

:INCCSP:10:

xinp1sel = csp, xinp2sel = d, cspalt = yes, next_mic = +;

to-page-zero = yes, xinp1sel = pc, xinp2sel = val4, next_pc = accux, next_mic =
0;



/*


Sonderbefehle
=============

%print

*/

:PRINT:60:

use_aluOp_h = yes, aluOp = printAddr, next_mic = +;

addr_sel = aluOut, datsel = tos, writemem = yes, next_mic = +;

to-page-zero = yes, xinp1sel = pc, xinp2sel = val4, next_pc = accux, next_mic =
0;


/*
```

```
%svref
%set-svref

*/

:svref:22:


alusource = val12, aluOp = plusAddr, use_aluOp_h = yes, next_mic = +;
datsel = aluOut, datinsel = tos, next_mic = +;
addr_sel = dsp, readmem = yes, xinp1sel = dsp, xinp2sel = val4, dspalt = yes, ne
xt_mic = +;
datsel = datain, alusource = databus, aluOp = plusAddr, next_mic = +;
addr_sel = aluOut, readmem = yes, next_mic = +;
datsel = datain, datinsel = tos, xinp1sel = pc, xinp2sel = val4, next_pc = accux
, to-page-zero = yes, next_mic = 0;


:set-svref:24:

/* save tos to regB for later use in assignment */
datsel = tos, datinsel = regB, next_mic = +;


/* start compute assignment address */
/* read middle arg */
addr_sel = dsp, readmem = yes, xinp1sel = dsp, xinp2sel = val4, dspalt = yes, ne
xt_mic = +;

datsel = datain, datinsel = tos, next_mic = +;

alusource = val12, aluOp = plusAddr, use_aluOp_h = yes, next_mic = +;

datsel = aluOut, datinsel = tos, next_mic = +;

/* read lowest arg */
addr_sel = dsp, xinp1sel = dsp, xinp2sel = val4, dspalt = yes, readmem = yes, ne
xt_mic = +;

datsel = datain, alusource = databus, aluOp = plusAddr, use_aluOp_h = yes, next_
mic = +;

/* write old top arg in regB to computed address */
addr_sel = aluOut, datsel = regB, writemem = yes, next_mic = +;

/* write assigned val to tos */
datsel = regB, datinsel = tos, to-page-zero = yes, xinp1sel = pc, xinp2sel = val
4, next_pc = accux, next_mic = 0;



:closure-ref:26:

/* compute read address as tos + 8 */
alusource = val8, aluOp = plusAddr, use_aluOp_h = yes, next_mic = +;
datsel = aluOut, datinsel = tos, next_mic = +;
/* read middle arg */
addr_sel = dsp, readmem = yes, xinp1sel = dsp, xinp2sel = val4, dspalt = yes, ne
xt_mic = +;
datsel = datain, alusource = databus, aluOp = plusAddr, next_mic = +;
/* read from computed address */
addr_sel = aluOut, readmem = yes, next_mic = +;
```



```
datsel = datain, datinsel = tos, xinp1sel = pc, xinp2sel = val4, next_pc = accux
, to-page-zero = yes, next_mic = 0;



/*


%closure-ref

*/



/*

%symbol-function
%set-symbol-function

*/

/*

symbol-functions

*/


:symbol-function:36:

alusource = val12, aluOp = plusAddr, use_aluOp_h = yes, next_mic = +;

addr_sel = aluOut, readmem = yes, next_mic = +;

datsel = datain, datinsel = tos, to-page-zero = yes, xinp1sel = pc, xinp2sel = v
al4, next_pc = accux, next_mic = 0;




:set-symbol-function:37:

/* save value to set in regB */
datsel = tos, datinsel = regB, next_mic = +;

/* read middle arg, advance dsp to lowest arg */
addr_sel = dsp, xinp1sel = dsp, xinp2sel = val4, dspalt = yes, readmem = yes, ne
xt_mic = +;

datsel = datain, datinsel = tos, next_mic = +;

alusource = val12, aluOp = plusAddr, use_aluOp_h = yes, next_mic = +;

/* set computed address value to regB */
addr_sel = aluOut, datsel = regB, writemem = yes, next_mic = +;

/* set tos to regB */
datsel = regB, datinsel = tos, to-page-zero = yes, xinp1sel = pc, xinp2sel = val
4, next_pc = accux, next_mic = 0;


/*
```

```
symbol-values

%symbol-value
%set-symbol-value

*/

:symbol-value:38:

alusource = val8, aluOp = plusAddr, use_aluOp_h = yes, next_mic = +;

addr_sel = aluOut, readmem = yes, next_mic = +;

datsel = datain, datinsel = tos, to-page-zero = yes, xinp1sel = pc, xinp2sel = v
al4, next_pc = accux, next_mic = 0;



:set-symbol-value:39:

/* save value to set in regB */
datsel = tos, datinsel = regB, next_mic = +;

/* read middle arg, advance dsp to lowest arg */
addr_sel = dsp, xinp1sel = dsp, xinp2sel = val4, dspalt = yes, readmem = yes, ne
xt_mic = +;

datsel = datain, datinsel = tos, next_mic = +;

alusource = val8, aluOp = plusAddr, use_aluOp_h = yes, next_mic = +;

/* set computed address value to regB */
addr_sel = aluOut, datsel = regB, writemem = yes, next_mic = +;

/* set tos to regB */
datsel = regB, datinsel = tos, to-page-zero = yes, xinp1sel = pc, xinp2sel = val
4, next_pc = accux, next_mic = 0;




/*

Statuswort-Operationen
======================
*/


:get-status:42:

addr_sel = dsp_m4, datsel = tos, xinp1sel = dsp, xinp2sel = val_4, dspalt = yes,
 writemem = yes, next_mic = +;

datsel = status, datinsel = tos, to-page-zero = yes, xinp1sel = pc, xinp2sel = v
al4, next_pc = accux, next_mic = 0;



:set-status:43:

set_status = yes, to-page-zero = yes, xinp1sel = pc, xinp2sel = val4, next_pc =
```



```
accux, next_mic = 0;


/*
DSP und CSP Operationen
=======================
*/


:get-dsp:44:

addr_sel = dsp_m4, datsel = tos, xinp1sel = dsp, xinp2sel = val_4, dspalt = yes,
 writemem = yes, next_mic = +;

datsel = dsp, datinsel = tos, to-page-zero = yes, xinp1sel = pc, xinp2sel = val4
, next_pc = accux, next_mic = 0;



:get-csp:45:

addr_sel = dsp_m4, datsel = tos, xinp1sel = dsp, xinp2sel = val_4, dspalt = yes,
 writemem = yes, next_mic = +;

datsel = csp, datinsel = tos, to-page-zero = yes, xinp1sel = pc, xinp2sel = val4
, next_pc = accux, next_mic = 0;




/*

Später zu implementierende Basisbefehle
=======================================


Elementare gets
===============

(defun get-byte (paddr offset)


*/

:get-byte:48:

addr_sel = dsp, readmem = yes, xinp1sel = dsp, xinp2sel = val4, dspalt = yes, ne
xt_mic = +;

datsel = datain, alusource = databus, aluOp = pAdd, use_aluOp_h = yes, next_mic
= +;

addr_sel = aluOut, readmem = yes, mtyp = byte, next_mic = +;

datsel = datain2, datinsel = tos, to-page-zero = yes, xinp1sel = pc, xinp2sel =
val4, next_pc = accux, next_mic = 0;


/*


(defun get-hword (paddr offset)
```

```
(defun get-word (paddr offset)

*/

:get-word:50:

addr_sel = dsp, readmem = yes, xinp1sel = dsp, xinp2sel = val4, dspalt = yes, ne
xt_mic = +;

datsel = datain, alusource = databus, aluOp = pAdd, use_aluOp_h = yes, next_mic
= +;

addr_sel = aluOut, readmem = yes, next_mic = +;

datsel = datain, datinsel = tos, to-page-zero = yes, xinp1sel = pc, xinp2sel = v
al4, next_pc = accux, next_mic = 0;


/*


Elementare sets
===============

(defun set-byte (paddr offset i)
(defun set-hword (paddr offset i)
(defun set-word (paddr offset pval)

*/

:set-byte:49:

/* save the value to set, tos, to regB  and read middle arg*/
datsel = tos, datinsel = regB, addr_sel = dsp, readmem = yes, next_mic = +;

/* compute address, middle arg to tos */
datsel = datain, datinsel = tos, xinp1sel = dsp, xinp2sel = val4, dspalt = yes,
next_mic = +;

/* read lowest arg */
addr_sel = dsp, readmem = yes, xinp1sel = dsp, xinp2sel = val4, dspalt = yes, ne
xt_mic = +;

/* do the address computation with middle and lowest arg */
datsel = datain, alusource = databus, aluOp = pAdd, use_aluOp_h = yes, next_mic
= +;

/* write out saved regB to computed address */
addr_sel = aluOut, datsel = regB2, writemem = yes, mtyp = byte, next_mic = +;

/* put regB into tos too */
datsel = regB, datinsel = tos, to-page-zero = yes, xinp1sel = pc, xinp2sel = val
4, next_pc = accux, next_mic = 0;




:set-word:51:

/* save the value to set, tos, to regB  and read middle arg*/
datsel = tos, datinsel = regB, addr_sel = dsp, readmem = yes, next_mic = +;
```



```
/* compute address, middle arg to tos */
datsel = datain, datinsel = tos, xinp1sel = dsp, xinp2sel = val4, dspalt = yes,
next_mic = +;

/* read lowest arg */
addr_sel = dsp, readmem = yes, xinp1sel = dsp, xinp2sel = val4, dspalt = yes, ne
xt_mic = +;

/* do the address computation with middle and lowest arg */
datsel = datain, alusource = databus, aluOp = pAdd, use_aluOp_h = yes, next_mic
= +;

/* write out saved regB to computed address */
addr_sel = aluOut, datsel = regB, writemem = yes, next_mic = +;

/* put regB into tos too */
datsel = regB, datinsel = tos, to-page-zero = yes, xinp1sel = pc, xinp2sel = val
4, next_pc = accux, next_mic = 0;


/*

car und cdr

*/


:CAR:28:

aluOp = get-addr, use_aluOp_h = yes, next_mic = +;

addr_sel = aluOut, readmem = yes, next_mic = +;

datsel = datain, datinsel = tos, to-page-zero = yes, xinp1sel = pc, xinp2sel = v
al4, next_pc = accux, next_mic = 0;


:CDR:29:

datinsel = regATo4, next_mic = +;

alusource = regA, aluOp = plusAddr, use_aluOp_h = yes, next_mic = +;

addr_sel = aluOut, readmem = yes, next_mic = +;

datsel = datain, datinsel = tos, to-page-zero = yes, xinp1sel = pc, xinp2sel = v
al4, next_pc = accux, next_mic = 0;


/*

fast transfer helpers

*/

:GETAINDEXPP:56:


/*
addro = dsp_m_4, datsource = tos, writemem = yes,
cdspsource = dsp_m_4, cdspdest = dsp_set_cdspbus, next_mic = +;
```

```
addro = regA, readmem = yes, next_mic = +;

datsource = datamem, datdest = tos, next_mic = +;

datdest = regA_inc_4, to-page-zero = yes, pcbsource = pc+4, next_mic = 0;
*/



:SETBINDEXPP:57:

/*
addro = regB, datsource = tos, writemem = yes, next_mic = +;

datdest = regB_inc_4, to-page-zero = yes, pcbsource = pc+4, next_mic = 0;

*/



:SETA:58:

/*
datsource = tos, datdest = regA,
to-page-zero = yes, pcbsource = pc+4, next_mic = 0;

*/

/*

:SETB:59:

datsource = tos, datdest = regB,
to-page-zero = yes, pcbsource = pc+4, next_mic = 0;

*/





/*

Unsichere Funktionen
====================

(defun p-constr (v-high v-low)

(defun p-add (p i)
(defun p-shiftl (p i)
(defun p-shiftr (n i)
(defun p-align (x i)

Tag-Funktionen
==============


(defun tag-and (p tag-code)
(defun tag-andc1 (p tag-code)
(defun tag-ior (p tag-code)
```



```
(defun tag-xor (p tag-code)
(defun tag-not (p)
(defun tag-eq (p tag-code)
(defun get-tag (p)
(defun tag-set (p tag-code)

Addr-Funktionen
===============

(defun addr-eq (p1 p2) *
(defun addr-eqi (p1 i) *
(defun addr-and (p i) *
(defun addr-andc1 (p i) *
(defun addr-ior (p i) *
(defun get-addr (p) *


Vergleich von Pointern
======================

(defun eq (x y) *


Arithmetische Funktionen
========================

(defun %+ (x y)
(defun %- (x y)

(defun %* (x y)
(defun %div (x y)

(defun %/= (x y)
(defun %= (x y)
(defun %> (x y)
(defun %< (x y)
(defun %<= (x y)
(defun %>= (x y)

*/

:ALUOPS:32:


addr_sel = dsp, readmem = yes, xinp1sel = dsp, xinp2sel = val4, dspalt = yes,  n
ext_mic = +;

datsel = datain, alusource = databus, next_mic = +;

datsel = aluOut, datinsel = tos, to-page-zero = yes, xinp1sel = pc, xinp2sel = v
al4, next_pc = accux, next_mic = 0;


:ALUOPS1:33:

next_mic = +;

datsel = aluOut, datinsel = tos, to-page-zero = yes, xinp1sel = pc, xinp2sel = v
al4, next_pc = accux, next_mic = 0;
```

```
:INITCPU:62:
datinsel = regATo0, next_mic = +;
addr_sel = regA, readmem = yes, next_mic = +;
datsel = datain, datinsel = regC, next_mic = +;
datinsel = regATo4, next_mic = +;
addr_sel = regA, readmem = yes, next_mic = +;
datsel = datain, datinsel = tp, next_mic = +;
datsel = tp, datinsel = regA, next_mic = +;
alusource = regA, use_aluOp_h = yes, aluOp = getLitBaseAddr, next_mic = +;
datsel = aluOut, datinsel = litbase, next_mic = +;
alusource = regA, use_aluOp_h = yes, aluOp = getTpCodeStart, next_mic = +;
addr_sel = aluOut, readmem = yes, next_mic = +;
datsel = datain, next_pc = databus, next_mic = +;
datsel = regC, datinsel = tos, to-page-zero = yes, next_mic = 0;

end
```