klee@informatik.unibw-muenchen.de 9.3.99

MPROZ – Ein 16-Bit Minimalprozessor

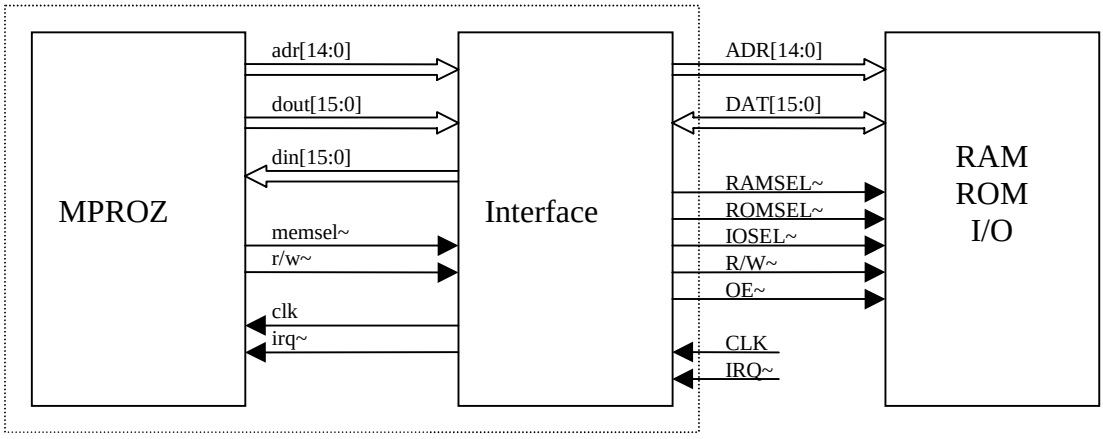
MPROZ ist eine vereinfachte Version des 16-Bit Prozessors XPROZ.

Die Schaltpläne (im WORKVIEW-Format) und der Assembler (QBasic) befinden sich in der Datei mproz.zip

1. Prozessorarchitektur

1.1 Schnittstellenbeschreibung des Prozessors

ADR[14:0]	: 15-Bit Adreßbus
DAT[15:0]	: 16-Bit Datenbus
RAMSEL~	: Zugriff auf RAM
ROMSEL~	: Zugriff auf ROM
IOSEL~	: Zugriff auf I/O
R/W~	: Schreib- / Lesezugriff
OE~	: Output Enable
CLK	: Takt
IRQ~	: Interrupt Request



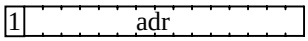
1.2 Register

Um den Hardwareaufwand so klein wie möglich zu halten, enthält das Programmiermodell von MPROZ neben einem 15-Bit Befehlszähler (PC) und einem 1-Bit Zustandsflag (F) keine weiteren Register. Nach einem Reset enthalten PC und F den Wert 0.

1.3 Befehlssatz

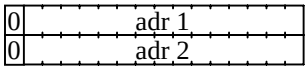
MPROZ unterstützt insgesamt drei Befehle:

br adr



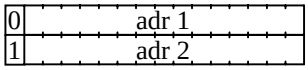
Lade adr in PC falls F=0
Lösche F

add adr1,adr2



Addiere den Inhalt von adr1 zu dem Inhalt von adr2 und speichere das Ergebnis in adr2.
Speichere das Übertragsbit der Addition in F.

nor adr1,adr2



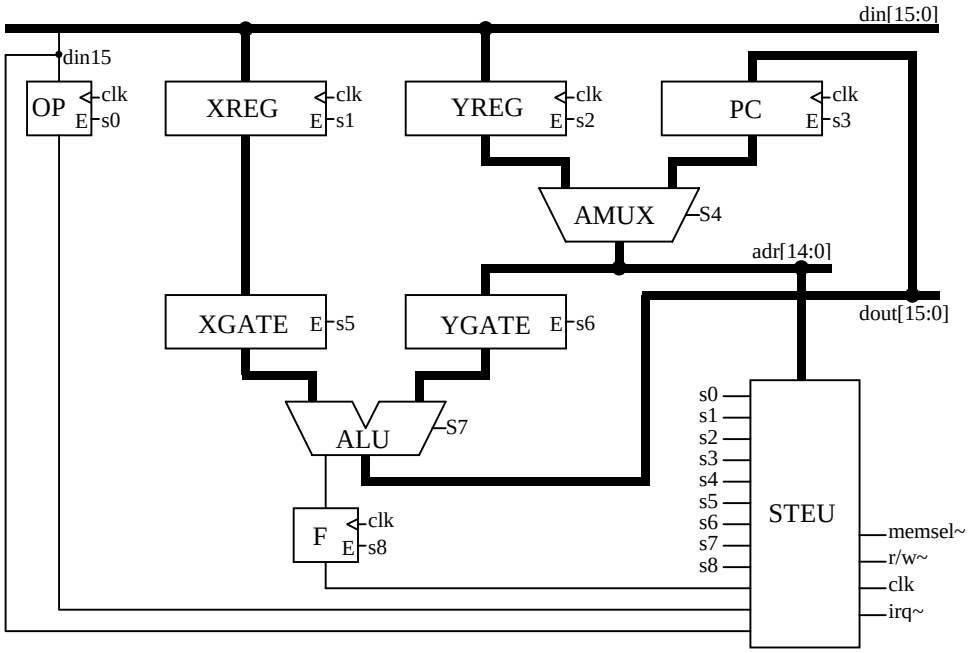
NOR-verknüpfe den Inhalt von adr1 mit dem Inhalt von adr2 und speichere das Ergebnis in adr2.
F=1 falls Ergebnis Null, sonst F=0.

1.4 Interrupt

Da es keinen eigenen Befehl zum Sperren und Freigeben des Interrupts gibt, wird das Interrupt Enable Flag (ie) durch einen Schreib- bzw. Lesezugriff auf die Adresse $4000 gelöscht bzw. gesetzt. Das Interrupt Request Signal (IRQ~) muß solange anliegen, bis der Interrupt bearbeitet wird. MPROZ bearbeitet einen Interrupt dann, wenn das Interrupt Enable Flag (ie) gesetzt ist und ein Sprungbefehl mit gelöschtem F-Flag (d.h. der Sprung würde ausgeführt) bearbeitet wird. Dadurch ist es nicht notwendig, das F-Flag und den Befehlszähler (PC) zu retten, sondern es genügt, den Sprungbefehl zu sichern. Dieser Sprungbefehl wird in der Speicherzelle gesichert, deren Adresse in Speicherzelle 1 steht (normalerweise Adresse $4000, da dann automatisch mit dem Sichern des Befehls das ie-Flag zurückgesetzt wird). Die Interruptroutine selbst beginnt an der Adresse 2. Die Rückkehr aus der Interruptroutine erfolgt durch einen Sprung zur Adresse $4000. Dort steht der gesicherte, durch den Interrupt unterbrochene Sprungbefehl, der nun erneut ausgeführt wird. Gleichzeitig wird durch das Lesen des Befehls aus Adresse $4000 auch das ie-Flag wieder gesetzt.

2. Implementierung

2.1 MPROZ



OP: 1-Bit Register
F: 1-Bit Register
PC: 15-Bit Register
XREG: 16-Bit Register
YREG: 16-Bit Register
AMUX: IF (s4=0) THEN out=in1 ELSE out=in2
XGATE: IF (s5=0) THEN out=$0000 ELSE out=in
YGATE: IF (s6=0) THEN out=$0001 ELSE out=in
ALU: IF (s7=0) THEN out=in1 ADD in2 , F=carry
ELSE out=in1 NOR in2 , F=zero
STEU: generiert Steuersignale s0-s8, memsel~ und r/w~

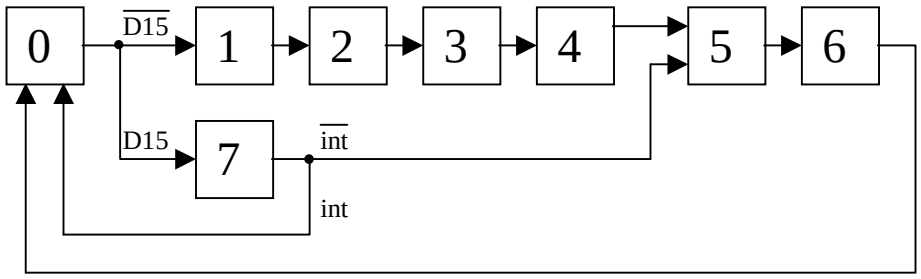
Zustand	Aktion
q0	PC→adr, PC+1→PC, din→XREG,YREG
q1	YREG→adr, din→XREG
q2	PC→adr, din→ YREG, din15→OP
q3	YREG→adr, din→YREG
q4	XREG [ADD|NOR] YREG → DOUT, [Carry|Zero] → F, DIN→XREG
q5	PC→adr, PC+1→PC, din→ YREG
q6	YREG→adr, XREG+0→dout
q7	[XREG | 1]+0→PC falls F=0, 0→F

int = irq~ + $\overline{ie}$ + F

Zustand	s8	s7	s6	s5	s4	s3	s2	s1	s0	memsel	r/w~
q0	0	0	1	0	1	1	1	1	-	0	1
q1	-	-	-	-	0	0	-	1	-	0	1
q2	-	-	-	-	1	0	1	0	1	0	1
q2*	-	0	1	0	1	1	1	0	1	0	1
q3	-	-	-	-	0	0	1	0	0	0	1
q4	1	OP	1	1	0	0	-	1	-	1	0
q5	0	0	1	0	1	1	1	0	-	0	1
q6	0	0	0	1	0	0	-	-	-	0	0
q7	1	0	0	int	-	$\overline{F}$	-	0	-	1	1

q2*: für 3-Adress-Befehle anstatt 2-Adress-Befehle

s0 = q2
s1 = q0 + q1 + q4
s2 = 1
s3 = q0 + q5 +q7 · $\overline{F}$
s4 = q0 + q2 + q5
s5 = q4 + q6 + q7 · int
s6 = $\overline{q6 + q7}$
s7 = q4 · OP
s8 = q4 + q7
memsel~ = q4 + q7
r/w~ = $\overline{q4 + q6}$



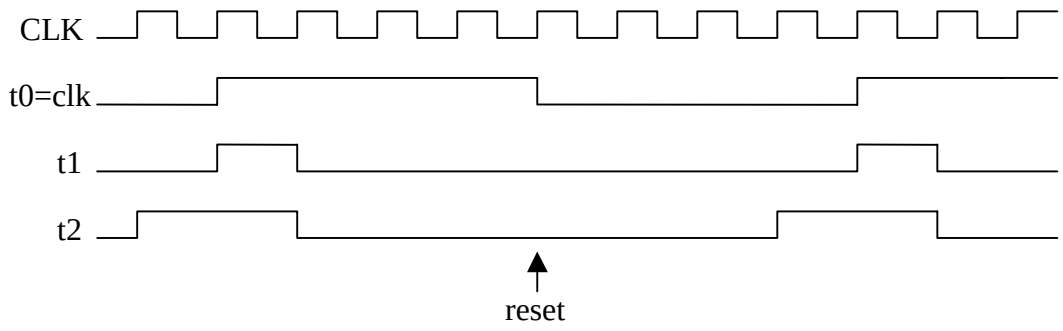
d0 = q6 + q7 · int
d1 = q0 · $\overline{D15}$
d2 = q1
d3 = q2
d4 = q3
d5 = q4 + q7 · $\overline{int}$
d6 = q5
d7 = q0 · D15

2.2 Interface

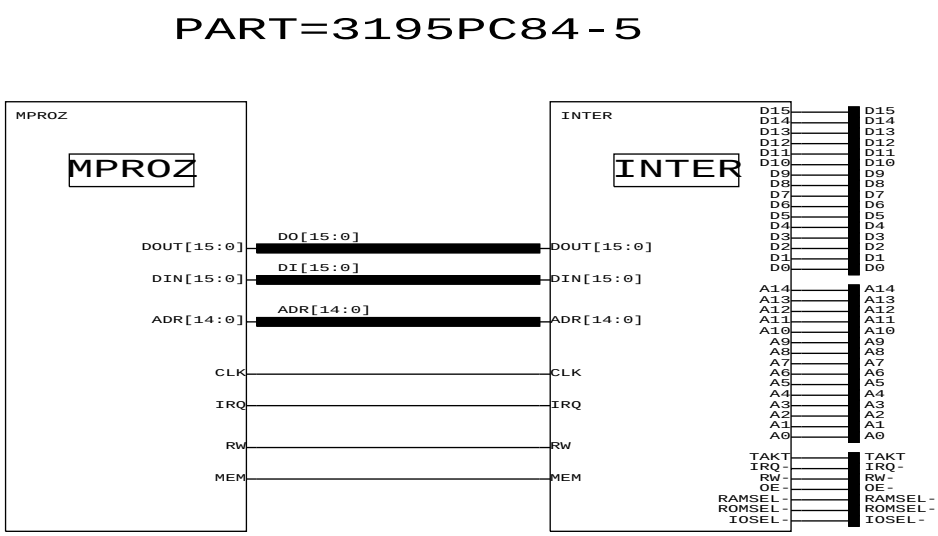
Das Interface ist zuständig für die Generierung der Ausgangssignale (RAMSEL~, ROMSEL~, IOSEL~, R/W~ und OE~).

Adreßbereich		
$0000 - $3fff	ROMSEL~	64 kByte
$4000 - $7dff	RAMSEL~	63 kByte
$7e00 - $7fff	IOSEL~	1 kByte

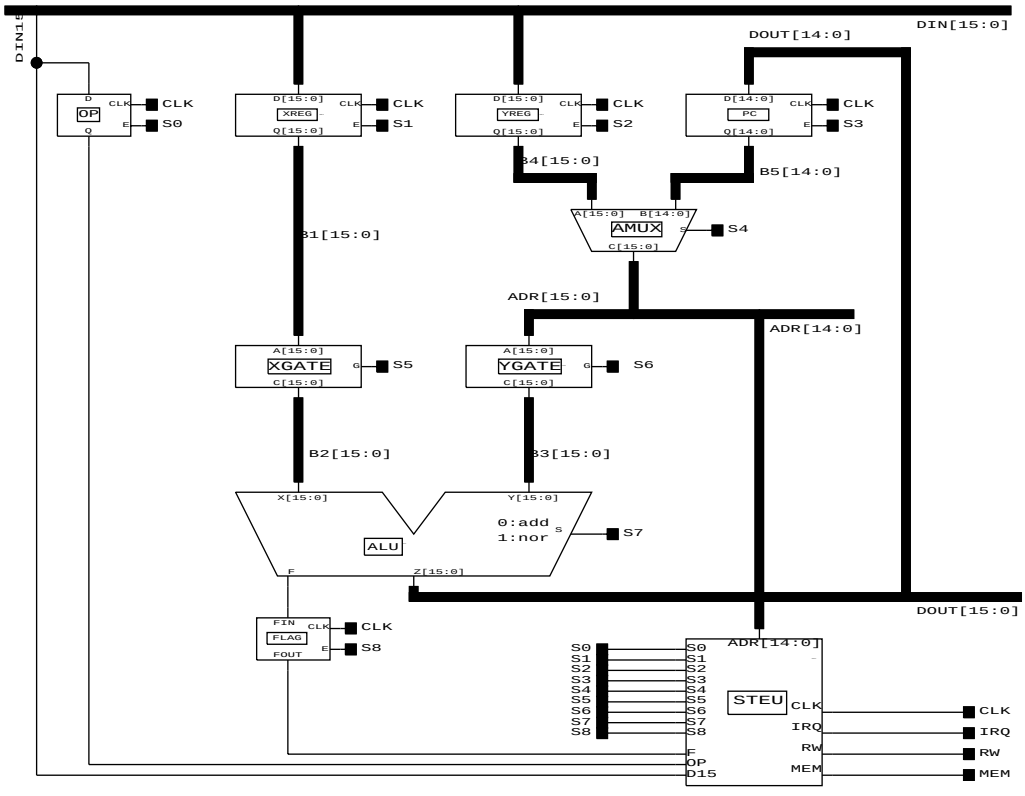
Für das exakte Timing wir das externe Taktsignal durch 8 geteilt.



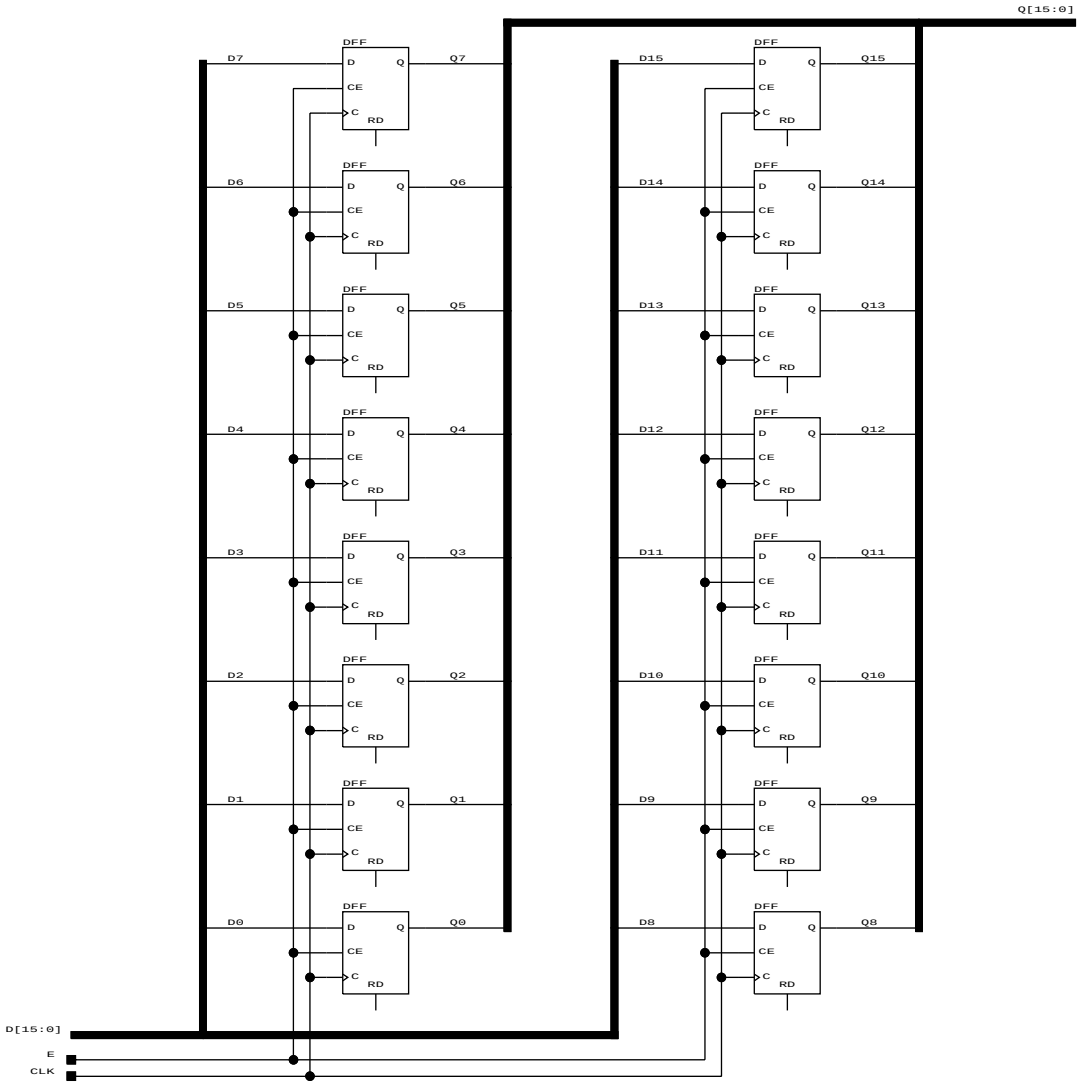
2.3 Schaltpläne



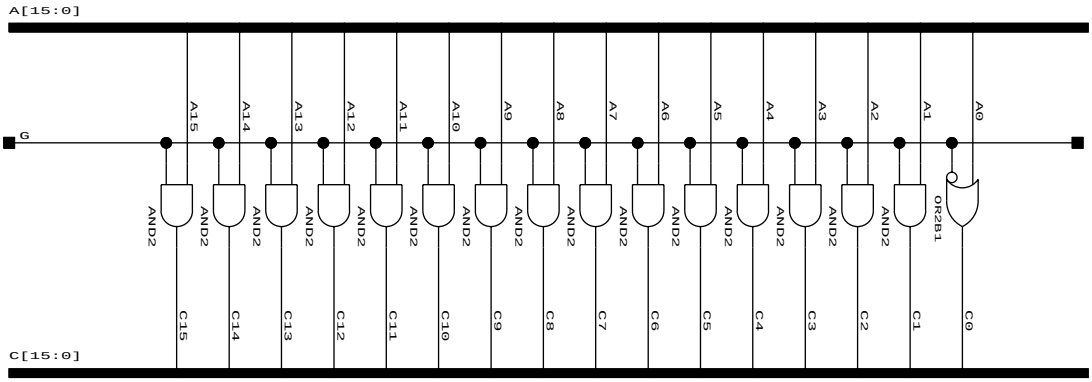
2.3.1 MPROZ



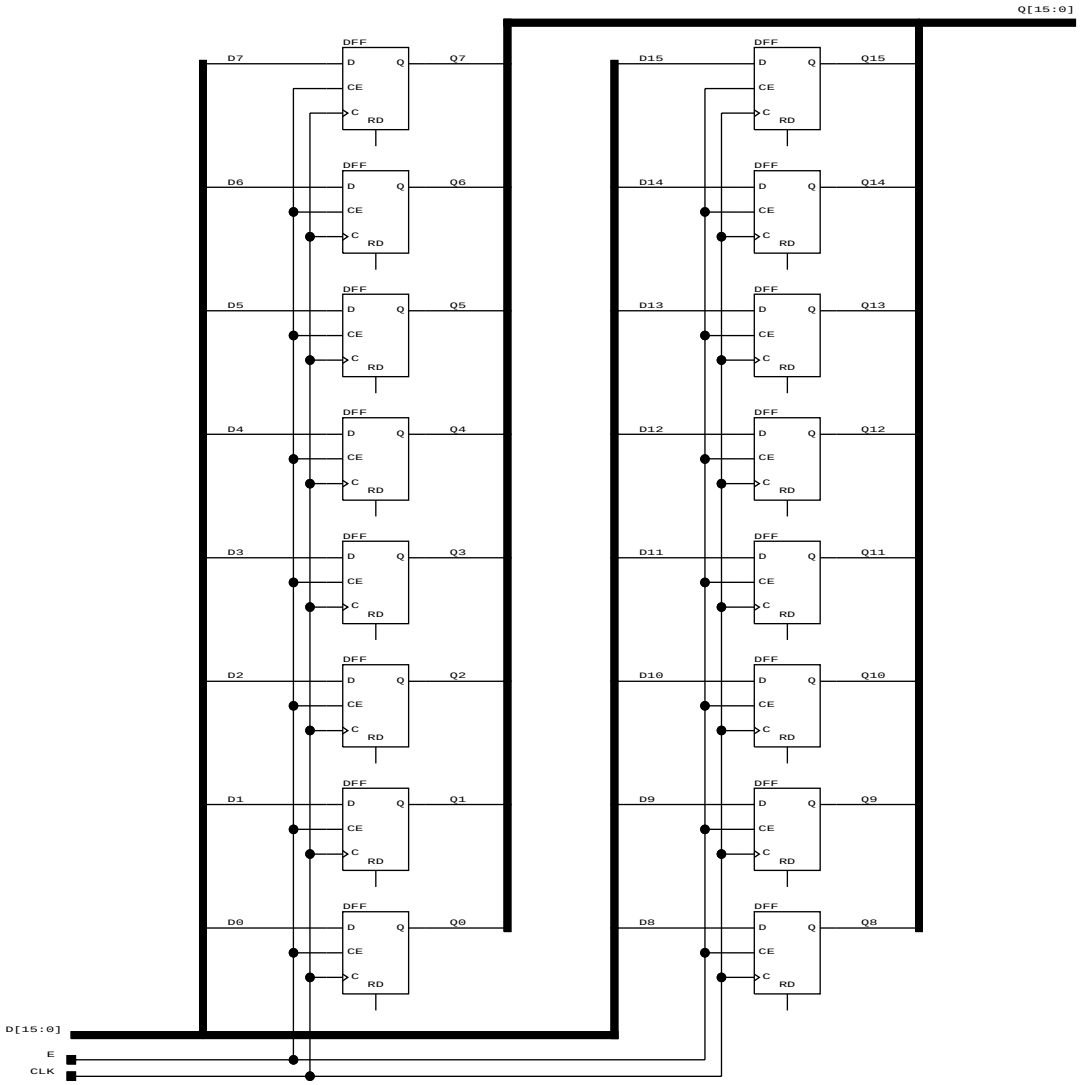
2.3.1.1 XREG



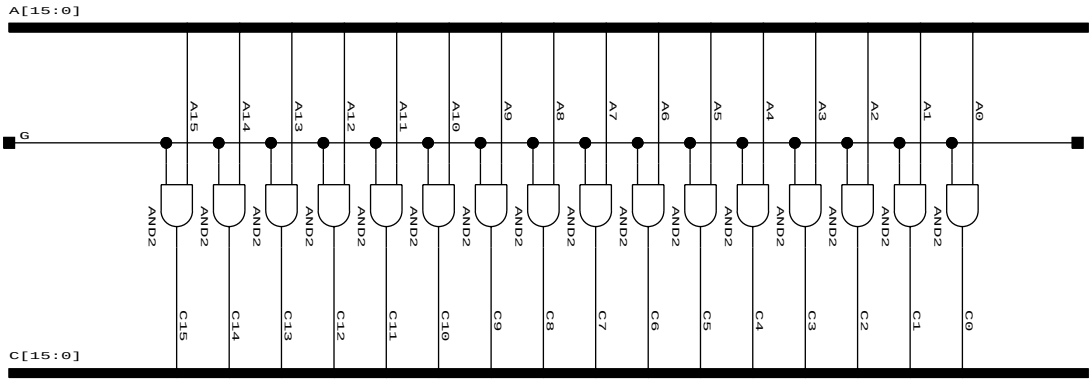
2.3.1.2 XGATE



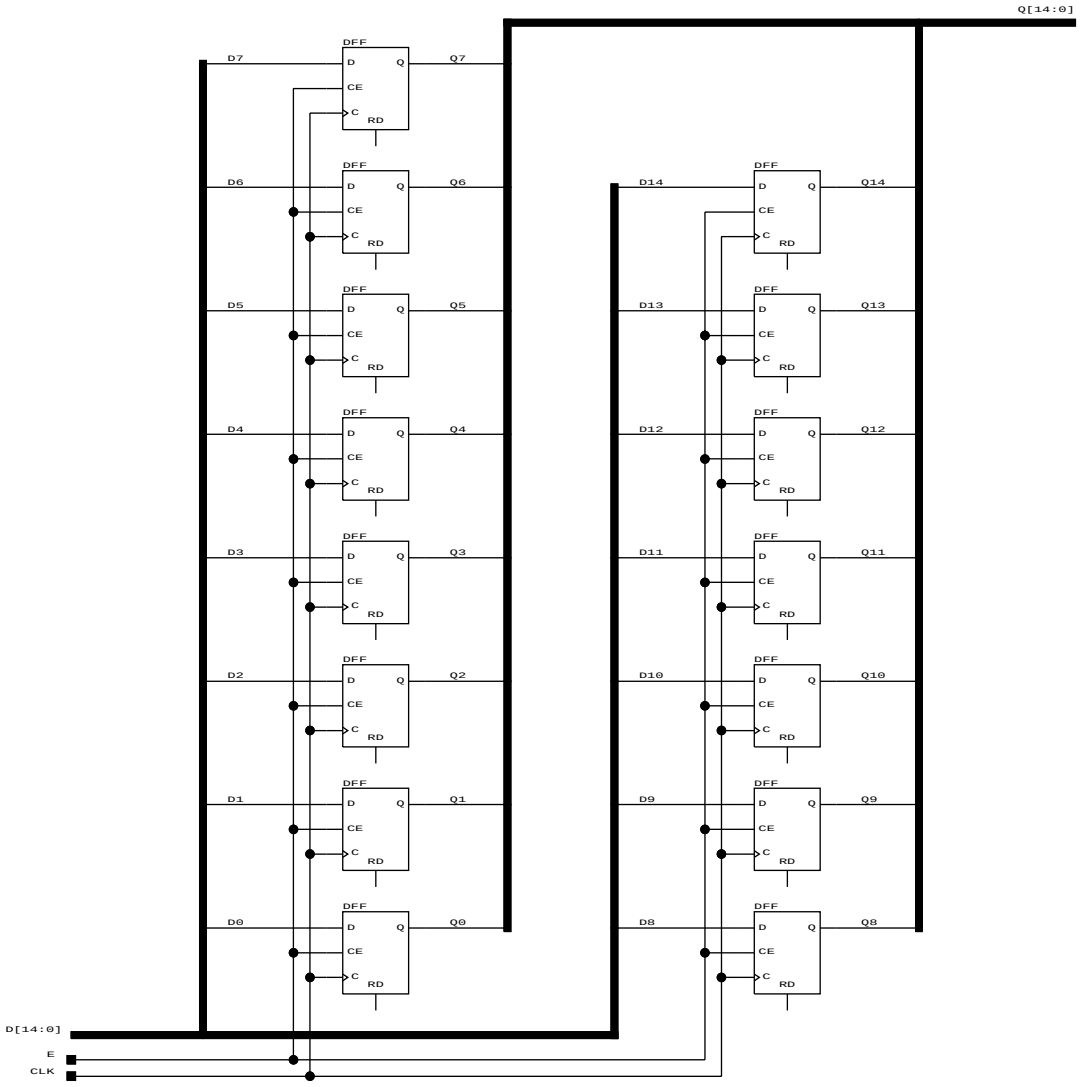
2.3.1.3 YREG



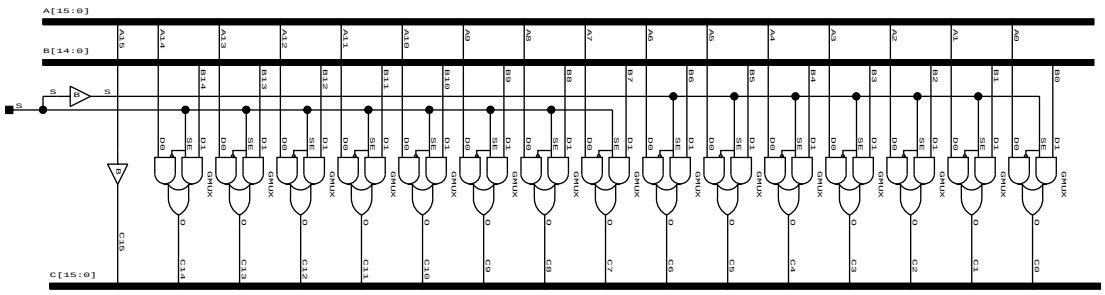
2.3.1.4 YGATE



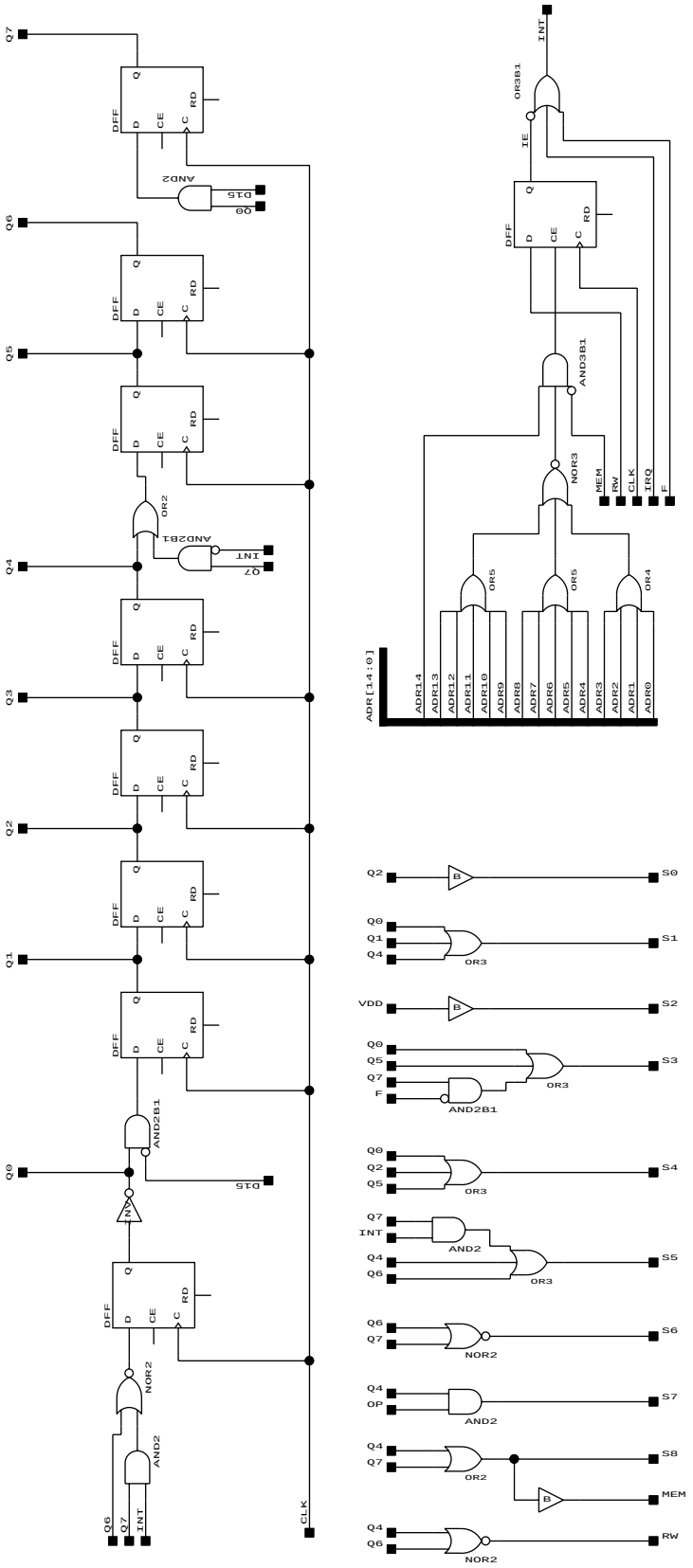
2.3.1.5 PC



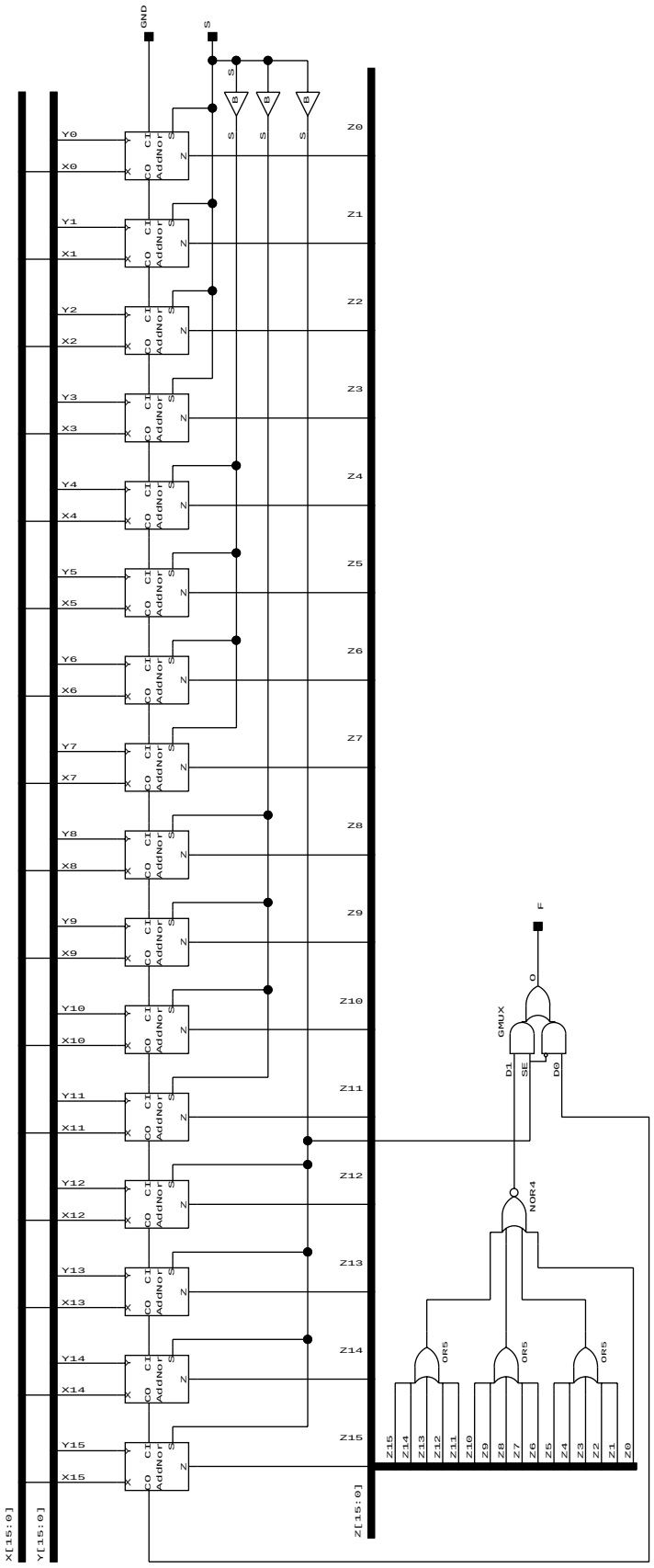
2.3.1.6 AMUX



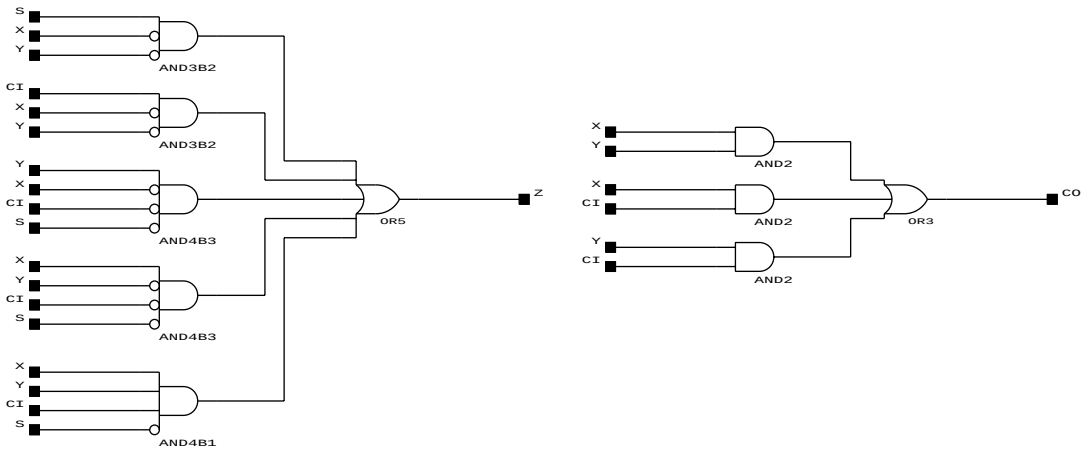
2.3.1.7 STEU


Q7
DFF
Q
D
CE
C
RD
AND2
D15
Q6
Q5
Q4
OR2
AND2B1
INT
Q7
Q3
Q2
Q1
Q0
INV
NOR2
AND2
CLK
INT
OR3B1
IE
AND3B1
NOR3
MEM
RW
CLK
IRQ
F
OR5
OR4
ADR[14:0]
ADR14
ADR13
ADR12
ADR11
ADR10
ADR9
ADR8
ADR7
ADR6
ADR5
ADR4
ADR3
ADR2
ADR1
ADR0
B
S0
S1
S2
S3
S4
S5
S6
S7
S8
OP

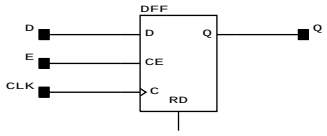
2.3.1.8 ALU


GND
S
S
S
S
S
B
B
B
Y0
X0
Y1
X1
Y2
X2
Y3
X3
Y4
X4
Y5
X5
Y6
X6
Y7
X7
Y8
X8
Y9
X9
Y10
X10
Y11
X11
Y12
X12
Y13
X13
Y14
X14
Y15
X15
Y
CI
CO
AddNor
S
N
X
Z0
Z1
Z2
Z3
Z4
Z5
Z6
Z7
Z8
Z9
Z10
Z11
Z12
Z13
Z14
Z15
X[15:0]
Y[15:0]
Z[15:0]
F
O
GMUX
D1
SE
D0
NOR4
OR5
OR5
OR5
Z15
Z14
Z13
Z12
Z11
Z10
Z9
Z8
Z7
Z6
Z5
Z4
Z3
Z2
Z1
Z0

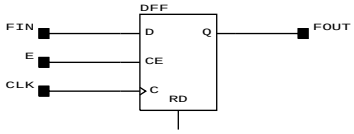
2.3.1.9 ADDNOR



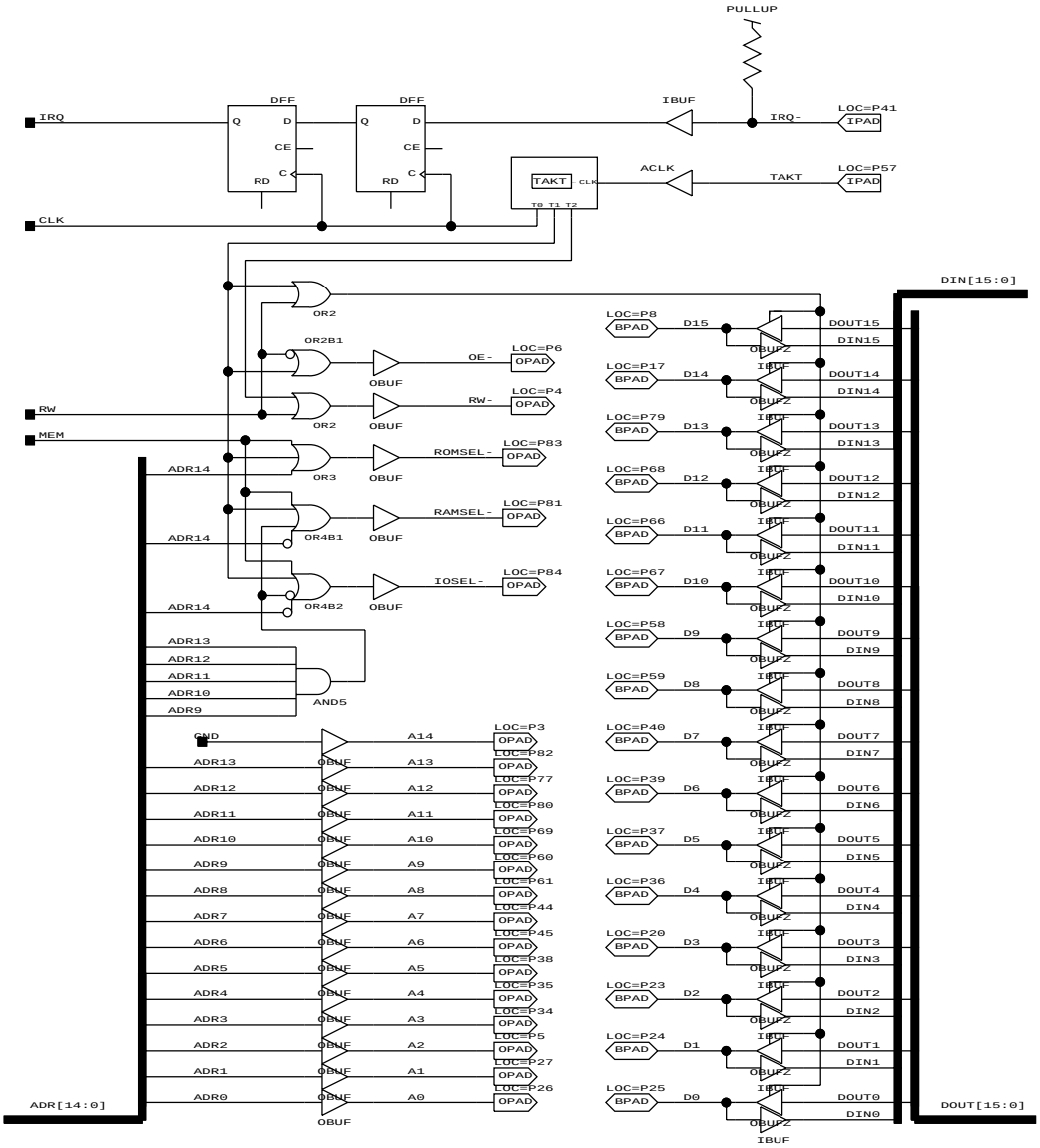
2.3.1.10 OP



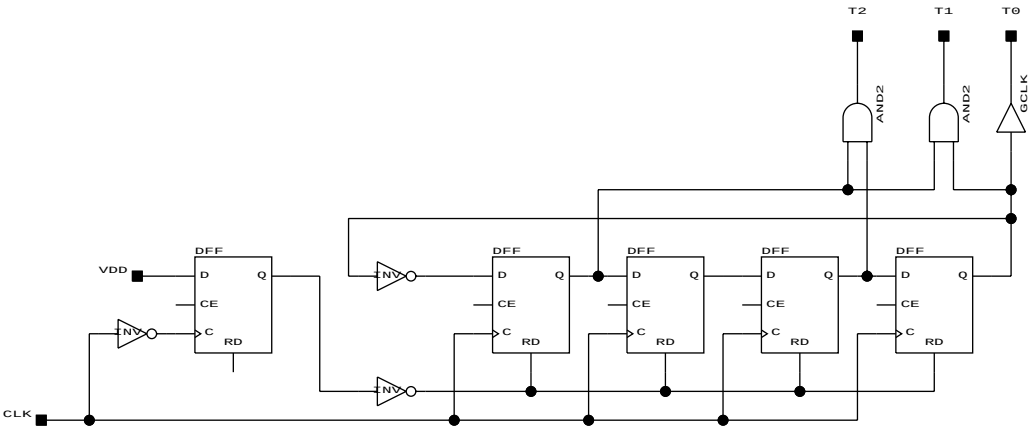
2.3.1.11 FLAG



2.3.2 Interface



2.3.2.1 TAKT



3. Assembler

Der Assembler unterstützt neben den drei realen Befehlen (add, nor, br) des MPROZ auch noch die Makrobefehle move und bsr (Unterprogrammaufruf) sowie die indirekte Adressierung. Zur besseren Lesbarkeit kann anstatt br auch bcc oder bne verwendet werden:

```
        add     r0,r1
        add     #20,r0
        add     (r0),r1
        add     r0,(r1)
        add     (r0),(r1)
        bcc     lab1            ; springe nach lab1 falls F=carry=0
        br      lab1            ; springe nach lab1 (da F jetzt auf alle Fälle =0)
lab2:   nor     r0,r1
        nor     #20,r0
        nor     (r0),r1
        nor     r0,(r1)
        nor     (r0),(r1)
        bne     lab1            ; springe nach lab1 falls F=zero=0
        br      lab1            ; springe nach lab1 (da F jetzt auf alle Fälle =0)
lab3:   move    r0,r1
        move    #20,r0
        move    (r0),r1
        move    r0,(r1)
        move    (r0),(r1)
        br      lab1            ; springe nach lab1 (da F =0 nach move)
        .
lab1:   bsr     sub
        .
        .
sub:    dcw     0       ; Speicherplatz für Rücksprungadresse
        .
        .
        .
        br      sub     ; Sprung zum Rücksprungbefehl

r0:             dc.w    0
r1:             dc.w    0
#20:            dc.w    20
#$ffff:         dc.w    #$ffff
#$7fff:         dc.w    #$7fff
```

Da diese Makros mittels selbstmodifizierendem Code realisiert sind, können sie nur bei Programmen im RAM nicht jedoch im ROM verwendet werden. Falls die Makros verwendet werden, müssen die beiden Zeilen

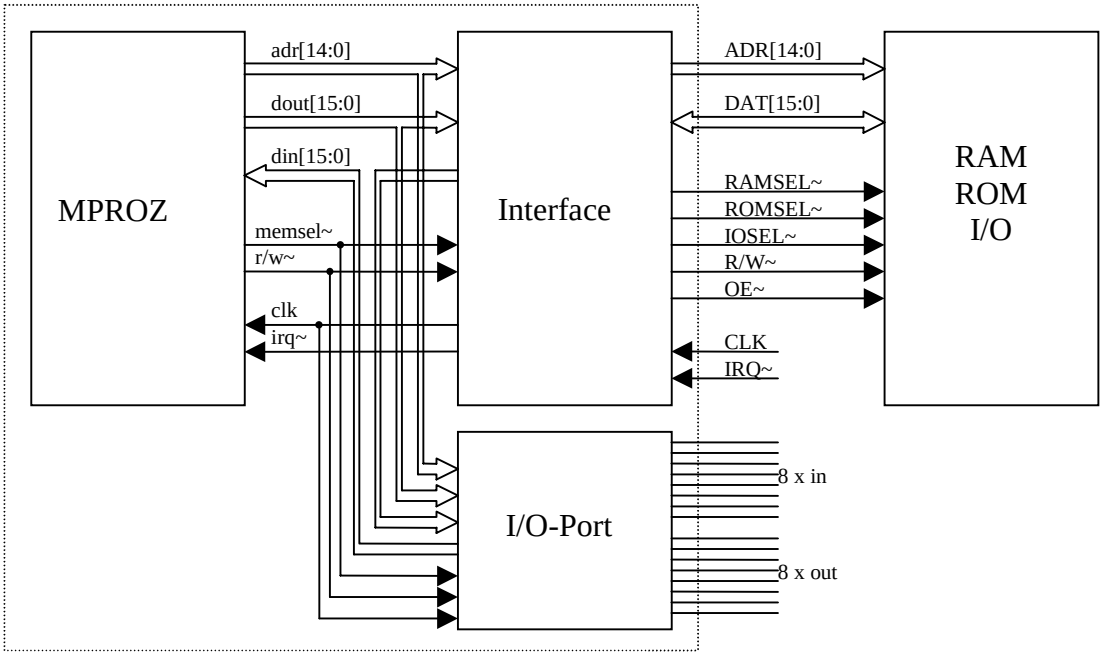
```
#$ffff:         dc.w    #$ffff
#$7fff:         dc.w    #$7fff
```

in das Programm eingefügt werden.

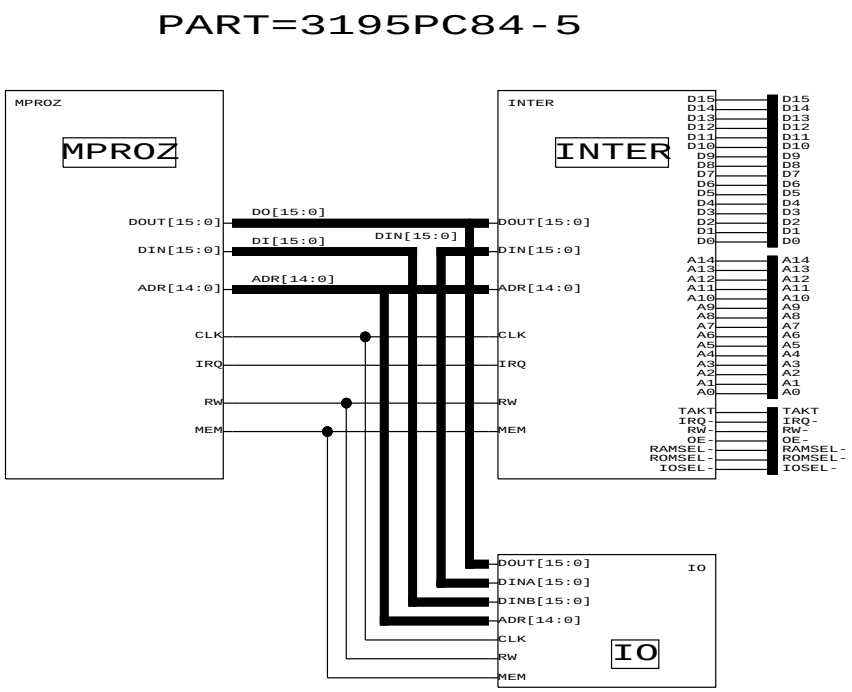
4. Erweiterungen

4.1 Ein- / Ausgabe

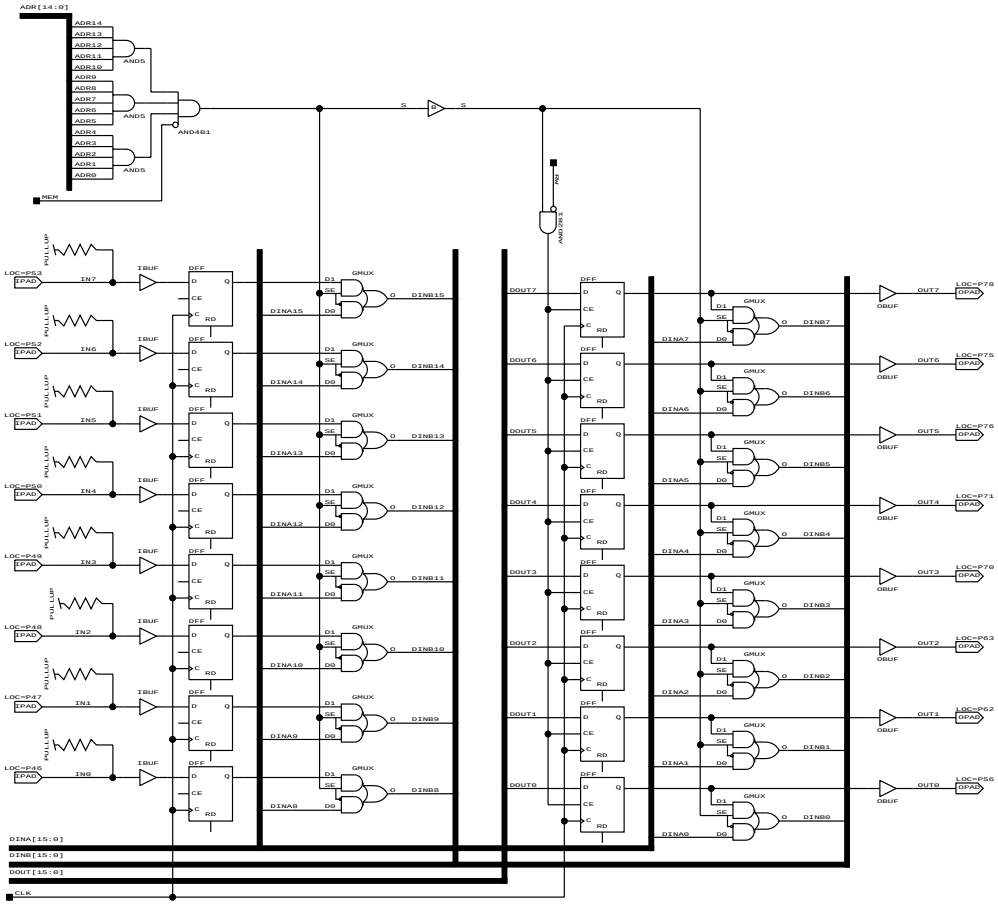
Um einfache Ein- /Ausgaben ohne zusätzliche Hardware machen zu können, wurde MPROZ um einen 8-Bit Eingangs- und einen 8-Bit Ausgangsport erweitert. Dieser Port wird unter der Adresse $7fff angesprochen. Bit 7-0 sind den Ausgangs- und Bit 15-8 den Eingangsleitungen zugeordnet.



4.1.1 Schaltpläne



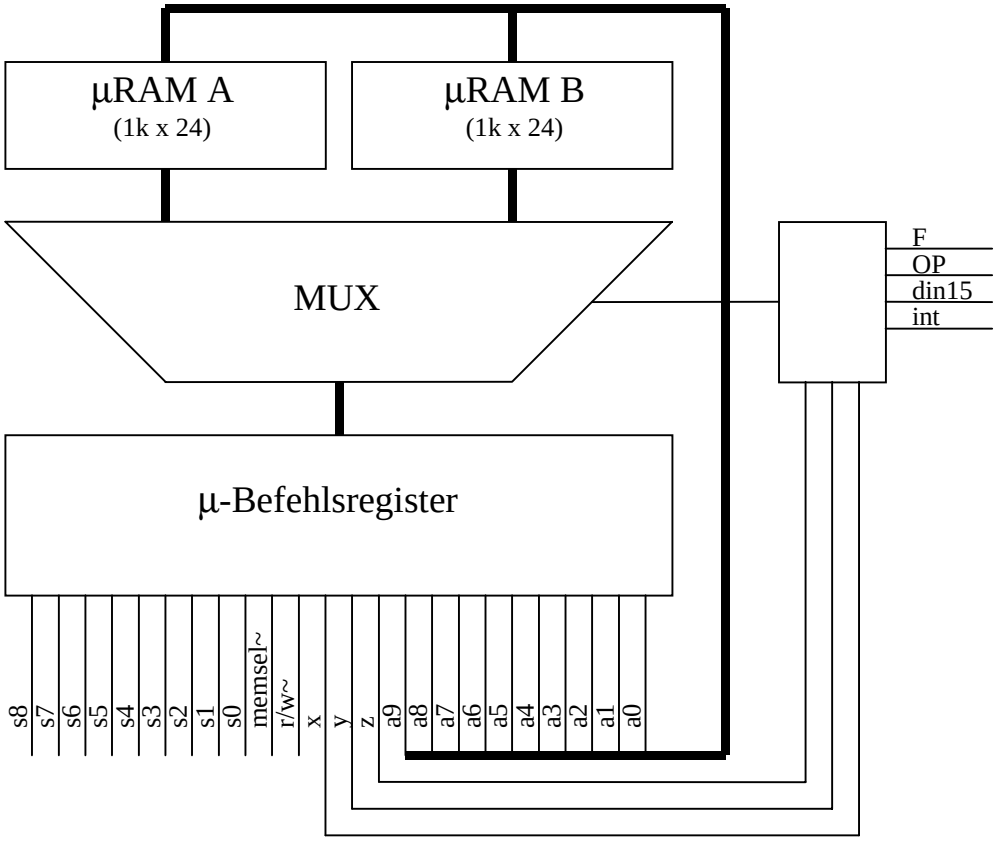
4.1.1 IO



4.2 Mikroprogrammierung

4.2.1 Hardware

Um das Prinzip der Mikroprogrammierung zu zeigen, wurde das Steuerwerk des MPROZ auch in einer mikroprogrammierten Version entworfen:



Die drei Bits x,y,z bestimmen dabei, ob das Mikrowort A oder B in das Mikrobefehlsregister übernommen wird:

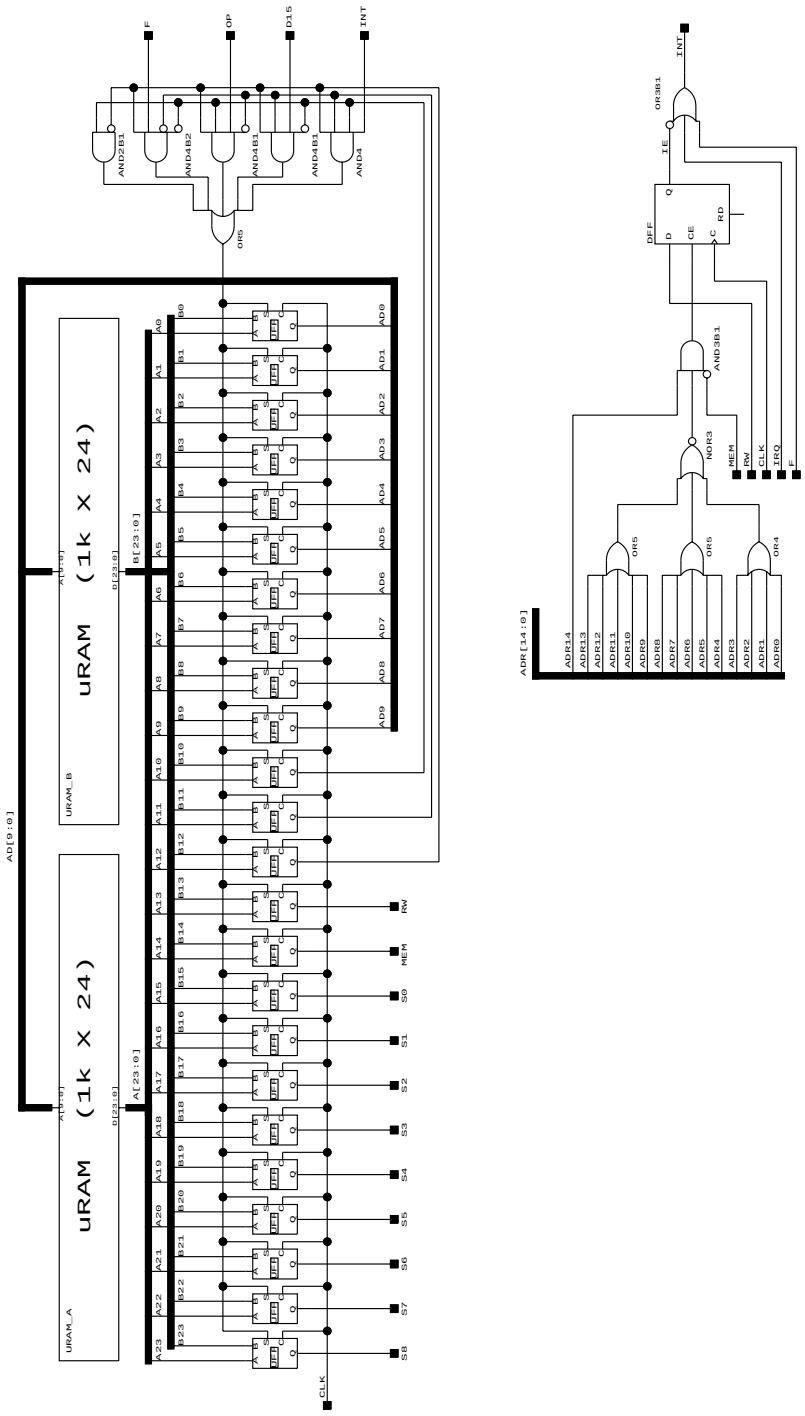
x y z	
0 - 0	A
0 - 1	B
1 0 0	A falls F=0 B falls F=1
1 0 1	A falls OP=0 B falls OP=1
1 1 0	A falls D15=0 B falls D15=1
1 1 1	A falls INT=0 B falls INT=1

4.2.1 Software

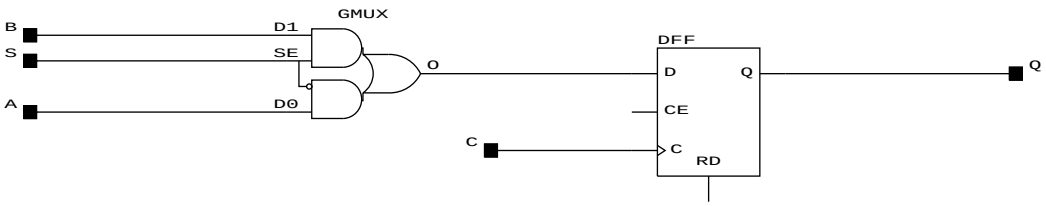
Das Mikroprogramm für das Steuerwerk des MPROZ:

	Mikrowort A							Mikrowort B								
Adresse	s8	s7	s6	s5	s4	s3	s2	s1	s0	mem	sre/lw	~	~	xyz	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0		
																									A	B
0	0010	1111	0011	1000	0000	0001	0010	1111	0011	1000	0000	0010	2f3801	2f3802
1	0000	0001	0010	0000	0000	0011	1001	0100	0111	1100	0000	0110	012003	947c06
2	0000	0001	0010	0000	0000	0011	1001	0000	0110	0000	0000	0000	012003	906000
3	0000	1010	1010	0100	0000	0011	0000	0010	0011	0100	0000	0100	0aa403	023404
4	1011	0001	0100	0000	0000	0101	1111	0001	0100	0000	0000	0101	b14005	f14005
5	0010	1110	0010	0100	0000	0101	0001	0000	0001	0000	0000	0000	2e2405	101000
6	1000	0100	0110	0000	0000	0101	0010	1111	0011	1000	0000	0001	846005	2f3801

4.2.3 Schaltpläne

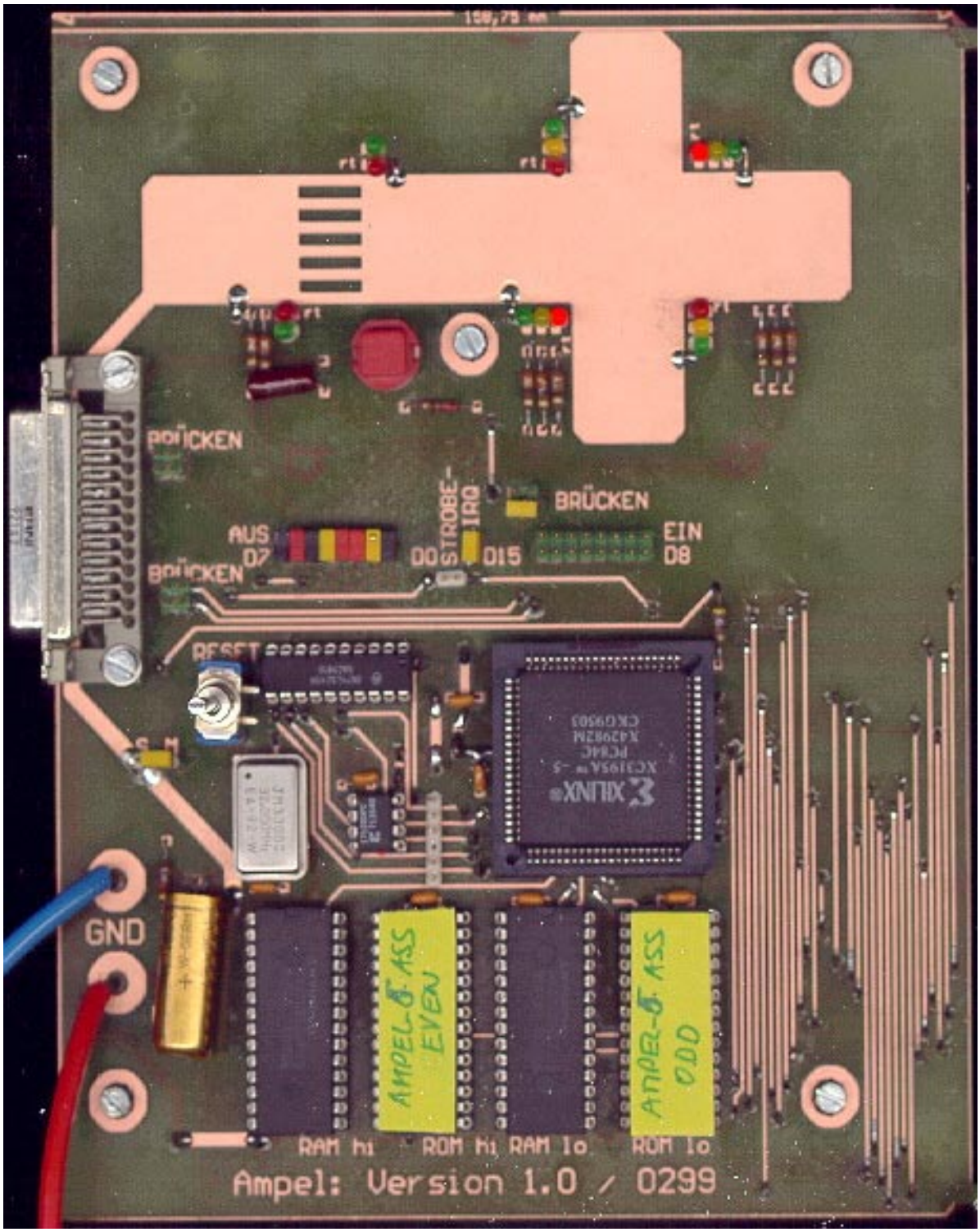


4.2.3.1 UFF



5. Test-Hardware

Um Funktionsfähigkeit des Prozessors zu zeigen, wurde eine kleine Schaltung (Ampelsteuerung) aufgebaut.



Das Programm für die Ampelsteuerung:

```
;******
;******    ampel_5.ass vom 10.März 1999    ******
;******    mit Fußgängerampel (Interrupt) ******
;
     pc=$0
        br      auto_amp
        dcw     $4000       ; Adresse zum retten des PC bei einem Interrupt
;
;****** Interruptroutine ab Adresse 0002 !!!    ******
;
;        br      irq         ; Adresse der Interruptroutine
irq:
        move    io,r0   ;
        nor     #ff3f,r0    ; Abfrage, ob FUSS AUS
        bne     set_fuss    ; FUSS AUS ---> FUSS auf rot
        move    io,r0
        add     #00c0,r0    ; FUSS auf ROT
        move    r0,io
set_fuss:
        br      $4000       ; Rücksprung aus Interrupt
        br      $4000       ; Rücksprung aus Interrupt
;
;********** Variablen **********
;
#$7fff: dcw     $7fff
#$ffff: dcw     $ffff
#$0001: dcw     $0001
#0000 : dcw     $0000
#0001 : dcw     $0001
;
;********** Die Zeitschleife "zeit" für 22 Sekunden hat im äußeren
;********** Schleifenregister den Wert $96 und im inneren den Wert
;********** $ffff bei einem 32MHz-Quarz (Teiler 8=250µsec)
;
#02s  : dcw     $0001       ; Zeitwert für 0,2 Sekunden
#05s  : dcw     $0003       ; Zeitwert für 0,5 Sekunden
#1s   : dcw     $0007       ; Zeitwert für 1 Sekunden
#2s   : dcw     $000d       ; Zeitwert für 2 Sekunden
#4s   : dcw     $001b       ; Zeitwert für 4 Sekunden
#6s   : dcw     $0029       ; Zeitwert für 6 Sekunden
#8s   : dcw     $0036       ; Zeitwert für 8 Sekunden
#22s  : dcw     $0096       ; Zeitwert für 22 Sekunden
;
#0076: dcw  $0076
#00c0: dcw  $00c0
#00db: dcw  $00db
#00eb: dcw  $00eb
#00f1: dcw  $00f1
#00f5: dcw  $00f5
#00f6: dcw  $00f6
#00e3: dcw  $00e3
#005b :dcw  $005b
#ff00 :dcw  $ff00
#ff3f :dcw  $ff3f
#ffbf: dcw  $ffbf
#feff: dcw  $feff
#ff7f: dcw  $ff7f
#ffff: dcw  $ffff
;
;
;********** Programmstart und Initialisierung **********
;
auto_amp:   move    #ff00,io    ; Alle LEDs AN
            move    #1s,r3
            move    auto_ini1,zeit_ret
```

```
            br      zeit
auto_ini1:  dcw $8000+auto_ini2
auto_ini2:  move    #ffff,io   ; Alle LEDs AUS
            move    #1s,r3
            move    auto_ini3,zeit_ret
            br      zeit
auto_ini3:  dcw $8000+auto_ini3a

auto_ini3a: move    #1s,r6
            nor     #0000,r6
auto_ini4:  move    #0001,r2   ; LED's nacheinander AN
auto_ini5:  move    r2,r4
            nor     #0000,r4
            move    r4,io       ; Ausgabe
            move    #02s,r3
            move    auto_ini6,zeit_ret
            br      zeit
auto_ini6:  dcw $8000+auto_ini7
auto_ini7:  move    r2,r5
            add     r5,r2
            move    r2,r5
            nor     #feff,r5
            bne     auto_ini5
            add     #0001,r6
            bcc     auto_ini4
auto_ini8:  move    #ffff,io   ; Alle LEDs AUS
            move    #2s,r3
            move    auto_ini9,zeit_ret
            br      zeit
auto_ini9:  dcw $8000+auto_ini10

auto_ini10: add     $4000,tmp   ; Lesezugriff: Interrupt enable
;
;********** Ampelsteuerung **********
;
auto_1: ;aampel_1: rt    - 6 sec / aampel_2: gn    - 6 sec; Wert:$xxdb

        move    io,r4   ;
        nor     #ff3f,r4    ; FUSS maskieren
        move    #00db,r5
        nor     #0000,r5
        nor     r5,r4
        move    r4,io

        move    #6s,r3
        move    a1_s,zeit_ret
        br      zeit
a1_s:   dcw $8000+auto_2

auto_2: ;aampel_1: rt    - 2 sec / aampel_2: ge    - 2 sec; Wert:$xxeb

        move    io,r4   ;
        nor     #ff3f,r4    ; FUSS maskieren
        move    #00eb,r5
        nor     #0000,r5
        nor     r5,r4
        move    r4,io

        move    #2s,r3
        move    a2_s,zeit_ret
        br      zeit
a2_s:   dcw $8000+auto_3


auto_3: ;aampel_1: rt/ge - 2 sec / aampel_2: rt    - 2 sec; Wert:$xxf1

        move    io,r4   ;
```

```
        nor     #ff3f,r4    ; FUSS maskieren
        move    #00f1,r5
        nor     #0000,r5
        nor     r5,r4
        move    r4,io
        move    #2s,r3
        move    a3_s,zeit_ret
        br      zeit
a3_s:   dcw $8000+auto_4

auto_4: ;aampel_1: gn    - 6 sec / aampel_2: rt    - 6 sec; Wert:$xxf6

        move    io,r4   ;
        nor     #ffbf,r4    ; Abfrage, ob FUSS rot
        bne     a4_1        ; FUSS ROT?
        move    #00f6,io    ; Normalbetrieb
        br      a4_2
a4_1:   move    #0076,io    ; FUSS auf grün setzen!
a4_2:   move    #6s,r3
        move    a4_s,zeit_ret
        br      zeit
a4_s:   dcw $8000+auto_5

        move    #6s,r3
        move    a4_s,zeit_ret
        br      zeit
a4_s:   dcw $8000+auto_5

auto_5: ;aampel_1: ge    - 2 sec / aampel_2: rt    - 2 sec; Wert:$xxf5

        move    io,r4
        nor     #ff7f,r4    ; FUSS grün?
        bne     a5_1
        move    io,r4       ; FUSS nicht grün
        nor     #ff3f,r4
        move    #00f5,r5
        nor     #0000,r5
        nor     r5,r4
        move    r4,io
        br      a5_2
a5_1:   move    #00f5,io    ; FUSS AUS
a5_2:   move    #2s,r3
        move    a5_s,zeit_ret
        br      zeit
a5_s:   dcw $8000+auto_6

auto_6: ;aampel_1: rt    - 2 sec / aampel_2: rt/ge - 2 sec; Wert:$xxe3

        move    io,r4   ;
        nor     #ff3f,r4    ; FUSS maskieren
        move    #00e3,r5
        nor     #0000,r5
        nor     r5,r4
        move    r4,io
        move    #2s,r3
        move    a6_s,zeit_ret
        br      zeit
a6_s:   dcw $8000+auto_1

;************   Zeitschleife         *************
;************   Zählerwert in r3     ************

zeit:   nor #$ffff,r1
        nor #0000,r3
        add #$0001,r3
w1i:    add #$0001,r1
        bcc w1i
```

```
        add #$0001,r3
        bcc w1i
        br  zeit_ret

;************   Umschalten auf RAM   ************
;
     pc=$4000
        dcw     0       ; Speicherzelle zum retten des PC bei einem Interrupt


r0:     dcw     0
r1:     dcw     0
r2:     dcw     0
r3:     dcw     0
r4:     dcw     0
r5:     dcw     0
r6:     dcw     0
doio_ret:dcw    0
zeit_ret:dcw    0
tmp:    dcw     0
io=$7fff

;****** Die Fußgängerampel schaltet mit aampel_2 (Querrichtung)
;
;auto_1=01xx: ;aampel_1: rt    - 6 sec / aampel_2: gn    - 6 sec; Wert:$xxdb
;auto_2=02xx: ;aampel_1: rt    - 2 sec / aampel_2: ge    - 2 sec; Wert:$xxeb
;auto_3=03xx: ;aampel_1: rt/ge - 2 sec / aampel_2: rt    - 2 sec; Wert:$xxf1
;auto_4=04xx: ;aampel_1: gn    - 6 sec / aampel_2: rt    - 6 sec; Wert:$xxf6
;auto_5=05xx: ;aampel_1: ge    - 2 sec / aampel_2: rt    - 2 sec; Wert:$xxf5
;auto_6=06xx: ;aampel_1: rt    - 2 sec / aampel_2: rt/ge - 2 sec; Wert:$xxe3
;fuss_1=07xx: ;fussampel: rot  - Beginn in auto_3 bis auto_6
;fuss_2=08xx: ;fussampel: grün - 8 sec in Phase auto_1 und auto_2

end:_
```