8BP3 Reference

Architecture

8BP3 is an 8-bit CPU with a 16-bit address bus. The currently implemented instructions are listed below. 8BP3 is a relatively standard processor with the exception that it has 256 I/O ports/registers. Some other features are interrupts (unimplemented) and a stack pointer (also not implemented). (Architecture written on 6/29/2019)

Internal Registers

Reg. 0 (Flags)
Read/Write

7	6	5	4	3	2	1	0
S	Z	C	H	V	P	D	E

Reg. 1 (Low byte of the stack pointer)
Read/Write

7	6	5	4	3	2	1	0
MSB							LSB

Reg. 2 (High byte of the stack pointer)
Read/Write

7	6	5	4	3	2	1	0
MSB							LSB

Reg. 3 (Low byte of the ISR's address)
Read/Write

7	6	5	4	3	2	1	0
MSB							LSB

Reg. 4 (High byte of the ISR's address)
Read/Write

7	6	5	4	3	2	1	0
MSB							LSB

Reg. 5 (Low byte of the program counter)
Read

7	6	5	4	3	2	1	0
MSB							LSB

Reg. 6 (High byte of the program counter)
Read

7	6	5	4	3	2	1	0
MSB							LSB

When reading from any of these registers, IN will not be asserted in order that an external register at the same address will not read onto the bus at the same time. When writing to these registers, it is possible to keep a copy of these registers in an external register because OUT is still pulsed. (Note 10/27/2017: Registers are mapped into I/O.)

Instruction Set

Arithmetic instructions

00 LDI I,R	# → B
01 MOV R,R	A → B
02 ST R,(RR+1)	A → (B)
03 LD (RR+1),R	(A) → B
04 ADD I,R,R	A + B → C
05 R,I,R	
06 R,R,R	
07 SUB I,R,R	A – B → C
08 R,I,R	
09 R,R,R	
0A AND I,R,R	A & B → C
0B R,I,R	
0C R,R,R	
0D OR I,R,R	A | B → C
0E R,I,R	
0F R,R,R	
10 XOR I,R,R	A ^ B → C
11 R,I,R	
12 R,R,R	
13 XNOR I,R,R	A ~^ B → C
14 R,I,R	
15 R,R,R	
16 MULT I,R,RR+1	A * B → C and C + 1 (Little endian)
17 R,I,RR+1	
18 R,R,RR+1	
19 DIV I,R,RR+1	A / B → C and C + 1 (Little endian)
1A R,I,RR+1	
1B R,R,RR+1	
1C ROL I,R,R	(A ≪ 1) + A[7] → B
1D R,I,R	
1E R,R,R	
1F ROR I,R,R	(A ≫ 1) + (A[0] ≪ 7) → B
20 R,I,R	
21 R,R,R	
22 ASL I,R,R	(A ≪ 1) + A[0] → B
23 R,I,R	
24 R,R,R	
25 ASR I,R,R	(A ≫ 1) + (A[7] ≪ 7) → B
26 R,I,R	
27 R,R,R	
28 INC I,R	A + 1 → B
29 R,R	
2A DEC I,R	A – 1 → B
2B R,R	
2C NOT I,R	~A → B
2D R,R	

2E NEG I,R	-A → B
2F R,R	
30 ADDC I,R,R	A + B + CF → C
31 R,I,R	
32 R,R,R	
33 SUBC I,R,R	A – B – CF → C
34 R,I,R	
35 R,R,R	
36 SETF I	A | FLAGS → FLAGS
37 R	
38 CLRF I	~A & FLAGS → FLAGS
39 R	
3A BCD I,R	BCD(A) → B
3B R,R	
3C BIN I,R	BIN(A) → B
3D R,R	
3E LDIW I,RR+1	# → B
3F MOVW RR+1,RR+1	A → B
40 STW RR+1,(RR+1)	A → (B)
41 LDW (RR+1),RR+1	(A) → B
42 STD R,II	A → (#)
43 STDW RR+1,II	
44 LDD II,R	(#) → A
45	
46	
47	
48	
49	
4A	
4B	
4C	
4D	
4E	
4F	
50	
51	
52	
53	
54	
55	
56	
57	
58	
59	
5A	
5B	
5C	
5D	

5E
5F
60
61
62
63
64
65
66
67
68
69
6A
6B
6C
6D
6E
6F
70
71
72
73
74
75
76
77
78
79
7A
7B
7C
7D
7E
7F

Branch instructions

80 CMP I,R	A – B, set flags
81 R,I	
82 R,R	
83 JMP II	A → PC
84 RR+1	
85 Jcc I,II	A → PC if CCs match flags
86 I,RR+1	
87 R,II	
88 R,RR+1	
89 JSR II	PC → (SP), A → PC
8A RR+1	
8B SRcc I,II	PC → (SP), A → PC if CCs match flags
8C I,RR+1	

8D	R,II	
8E	R,RR+1	
8F	RET	(SP) → PC
90	INT	If (EF == 1)? {0 → EF, PC → (SP), ILR → PC}
91	RETI	1 → EF, (SP) → PC
92	TEST R	Test for zero and sign.
93	INTcc I	If CCs match flags then INT
94		
95		
96		
97		
98		
99		
9A		
9B		
9C		
9D		
9E		
9F		
A0		
A1		
A2		
A3		
A4		
A5		
A6		
A7		
A8		
A9		
AA		
AB		
AC		
AD		
AE		
AF		
B0		
B1		
B2		
B3		
B4		
B5		
B6		
B7		
B8		
B9		
BA		
BB		
BC		

BD
BE
BF

Stack instructions

C0 POPB R (SP) → A
C1 (R)
C2 POPW RR+1 (SP) → A
C3 (RR+1)*
C4 POPF (SP) → FLAGS
C5 PSHB I A → (SP)
C6 R
C7 (R)
C8 PSHW II A → (SP)
C9 RR+1
CA (RR+1)*
CB PSHF FLAGS → (SP)
CC
CD
CE
CF
D0
D1
D2
D3
D4
D5
D6
D7
D8
D9
DA
DB
DC
DD
DE
DF
E0
E1
E2
E3
E4
E5
E6
E7
E8
E9
EA
EB

EC
ED
EE
EF
F0
F1
F2
F3
F4
F5
F6
F7
F8
F9
FA
FB
FC
FD
FE
FF

*The byte above the addressed byte is pushed onto the stack as well.
A red line indicates that this instruction has not been programmed into the control logic.

ALU Select Codes:
F(A,B?) =
00 A
01 B
02 A – B
03 –A
04 A + B
05 A & B
06 A | B
07 A ^ B
08 A ~^ B
09 A * B (Low byte)
0A A * B (High byte)
0B A % B
0C A / B
0D (A ≪ 1) + A[7]
0E (A ≫ 1) + (A[0] ≪ 7)
0F (A ≪ 1) + A[0]
10 (A ≫ 1) + (A[7] ≪ 7)
11 A + 1
12 A – 1
13 ~A
14 BCD(A)
15 BIN(A)

16 A + B + CARRY
17 A – B – CARRY

Instruction Description

ADCI #,rs,rd Add a constant plus the carry flag to a register. 6 cycles. 30,#,rs,rd

ANDI #, rs, rd AND a register with a constant and store in another register. 6 cycles. 0A,#,rs,rd

BR.cc ## If the flags satisfy the cc then branch to ##, else continue. 85,cc,#l,#h

BR.cc rs If the flags satisfy the cc then branch to {rs,rs+1}, else continue. 86,cc,rs

CMP rs1, rs2 Compare rs1 and rs2 (rs1 – rs2) and update flags. 82,rs1,rs2

CMPI rs, # Compare a register with a constant. 5 cycles. 81,rs,#

INC r Increment the specified register. Affects S, Z, P, C, H, and V flags. 29,r

JMP ## Jump to ##. 3 cycles. 83,#l,#h

LD rs, rd Load rd with the contents of the memory location pointed to by {rs+1, rs}. 03,rs,rd

LDD ##, rs Load direct to a register. 5 cycles. 44,#l,#h,rd

LDI #, rd Load a register with a constant. 3 cycles. 00,rd,#

MOV rs, rd Move the contents of the rs register to the rd register. 5 cycles. 01,rs,rd

ST rs, rd Store the contents of the rs register to the memory location pointed to by {rd+1, rd}. ST and LD are the only arithmetic and transfer instruction that allow storage and retrieval of data from memory. 02,rd,rs

STD rs, ## Store the contents of a register in memory. 5 cycles. 42,rs,#l,#h

Condition Codes

00 don't jump
01 jump
02 INTE
03 !INTE
04 DVZ
05 !DVZ
06 Even parity
07 Odd parity
08 OVR
09 !OVR
0a (C && HC)
0b !(C && HC)
0c (C || HC)

0d !(C || HC)
0e HC
0f !HC
10 C
11 !C
12 EQ/Z
13 !EQ/!Z
14 NEG
15 POS
16 don’t jump
17 jump
18 don’t jump
19 jump
1a don’t jump
1b jump
1c don’t jump
1d jump
1e don’t jump
1f jump