Appendix – Demonstration Processors

To validate the suitability of proposed on-line BISC-units and the micro rollback principle, different example processors were designed. The attention was focussed on COTS-processors. Designs should represent characteristics of embedded processor-cores in various applications. In order to cover a widespread similarity with real architectures, exemplary designs are redesigns – respective derivatives of microprocessors, micro-controller CPU-cores and digital signal-processors. The following sections of this appendix should give a short overview of used processors.

Circuits were in most cases designed in a mixed manner: logic-, RTL-design and synthesized VHDL-descriptions. As design environments, Viewlogic's WorkviewOffice – respective Innoveda's Electronic Design Center and SYNOPSYS hardware-compiler were mainly and CADENCE was partly used. Designs are available as schematic, EDIF-netlist or structural VHDL-description.

A.1 Microprocessor t4008

As an example for a simple 8-bit microprocessor class, the core t4008 was implemented. The top hierarchy of the design is built with the control- (CP) and the data-path (DP). As a ″von-Neumann″ architecture, the CP reads instructions and data from memory and controls the DP with the control-word CW.

M. Pflanz, On-line Error Detection and Fast Recover Techniques, LNCS 2270, pp. 87–115, 2002.

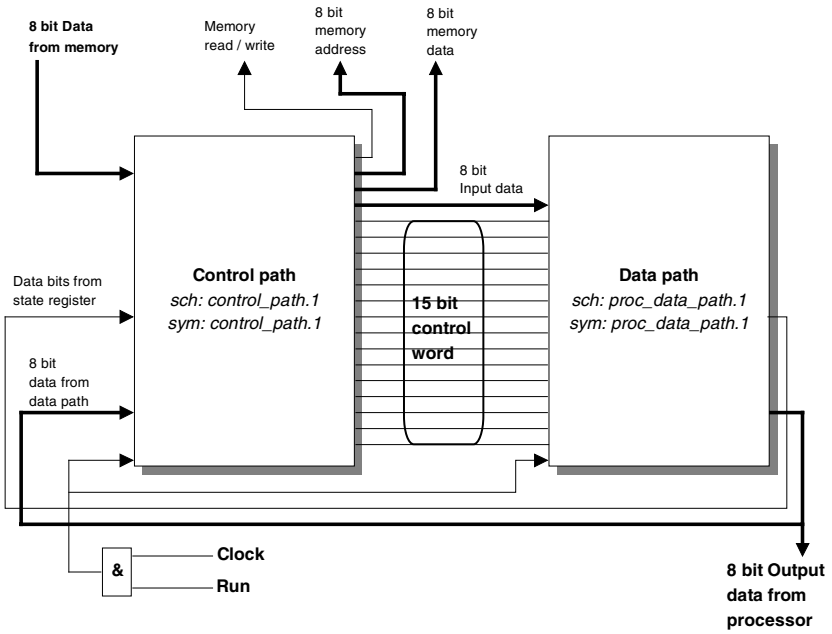
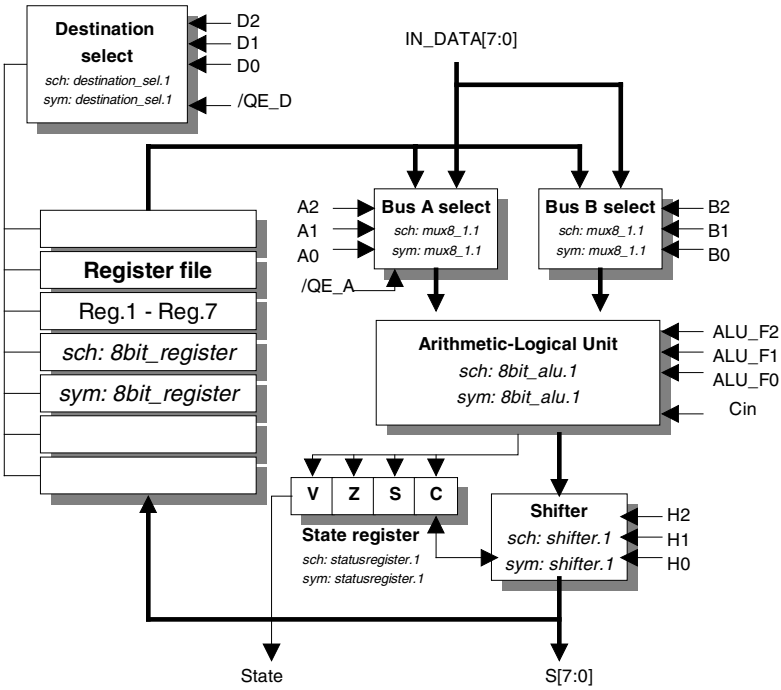


Fig. 57. Top-hierarchy view of the t4008

The DP contains an arithmetic-logical-unit (ALU) for common integer operations. The combinatorial shifter serves to carry out simple shift and rotate operations. A register-file with seven registers represents basic processor-registers. The flag-register is used for conditional instructions and ′rotate with carry′ operations. Input- and output-ports are directly connected to CP.



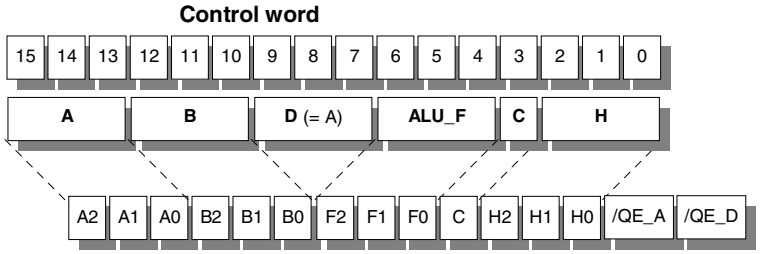


Fig. 58. Data-Path and Control-word Definition

Table 17. Destination Select

D[2:0] CW bits 9|8|7	load reg.
000	-
001	Reg. 1
010	Reg. 2
011	Reg. 3
100	Reg. 4
101	Reg. 5
110	Reg. 6
111	Reg. 7

Table 18. Source Select Signals

A_sel[2:0] CW bits 15|14|13	ALU A-INPUT	B_sel[2:0] CW bits 15|14|13	ALU B-INPUT
000	IN_DATA	000	IN_DATA
001	Reg. 1	001	Reg. 1
010	Reg. 2	010	Reg. 2
011	Reg. 3	011	Reg. 3
100	Reg. 4	100	Reg. 4
101	Reg. 5	101	Reg. 5
110	Reg. 6	110	Reg. 6
111	Reg. 7	111	Reg. 7

Table 19. ALU Function Select Signals

ALU_F[2:0], Cin CW bits 6|5|4|3	**Function ALU output**	**ALU_F[2:0], Cin CW bits 6|5|4|3**	**Function ALU output**
0000	F = A	100X	F = A OR B
0001	F = A+1	101X	F = A XOR B
0010	F = A+B	110X	F = A AND B
0011	F = A+B+1	111X	F = NOT A
0100	F = A-B-1		
0101	F = A-B		
0110	F = A-1		
0111	F = A; C←1		

Table 20. Shifter Control

Shifter control CW bits 2|1|0	**function**
000	transfer
001	Shift right
010	Shift left
011	Clear data
100	-
101	Rotate right
110	Rotate left
111	Clear data

The CP contains control- and buffer-registers. A hardwired control-logic generates control-signals and the CW for the DP according micro-instructions. An instruction flow of this *sequential* processor starts always with the fetch phase: The initialized program-counter PC is loaded into the memory-address register MAR. It addresses the first instruction within the memory. In the next cycle, the memory-buffer register

MBR is loaded with the first instruction-op-code. I will be moved to instruction-register IR, where the decoder generates the instruction variable for control-logic. After instruction decoding, different time-signal-controlled micro-instruction flows can be started.

The instruction set covers 61 instructions ($\iota_0, \iota_1, \ldots, \iota_{60}$) which are addressed with a 6-bit op-code (00_H to $3F_H$). The processor was specified as a *three-address-machine*, that means:

Destination register (A) ← Source register 1 (D) <operation> Source register 2 (B)

For a reduction of the effort for addressing, a *pseudo-two-address-mode* was introduced: Destination register A is equal to Source register D. The consequence of this mode is similar to the operation of processors with an accumulator.

The addressing is organized as follow:

with A=D:
Destination register (A) ← Source register(A) <operation> Source register(B)

This method needs an adaptation of the available move-instructions **<MVIN x>, <MOV x y>**, **<MVX x>** und **<MVM x m>**. Because the ALU can execute a data transfer without operation (A←A) only at bus-port A, the destination register must also be A. In order to realize a transfer to an other register, instead of an ALU-operation *transfer* A (F=011, C=1 or F=000, C=0), an operation *addition* of B and blocked A (bus-select = disable → all A-ALU-Inputs = 0) is executed: A ← 0 + B = A ← B.

In the DP, there are seven processor registers are available. They are addressable with 3-bit variables A and B (control-bits 15 to 10 at bus-selectors A and B; CW-bits 9, 8, 7 = D).

Table 21. t4008 Register addresses

000	Register INP or OUT
001	Register 1 (or a)
010	Register 2 (or b)
011	Register 3 (or c)
100	Register 4 (or d)
101	Register 5 (or e)
110	Register 6 (or f)
111	Register 7 (or g)

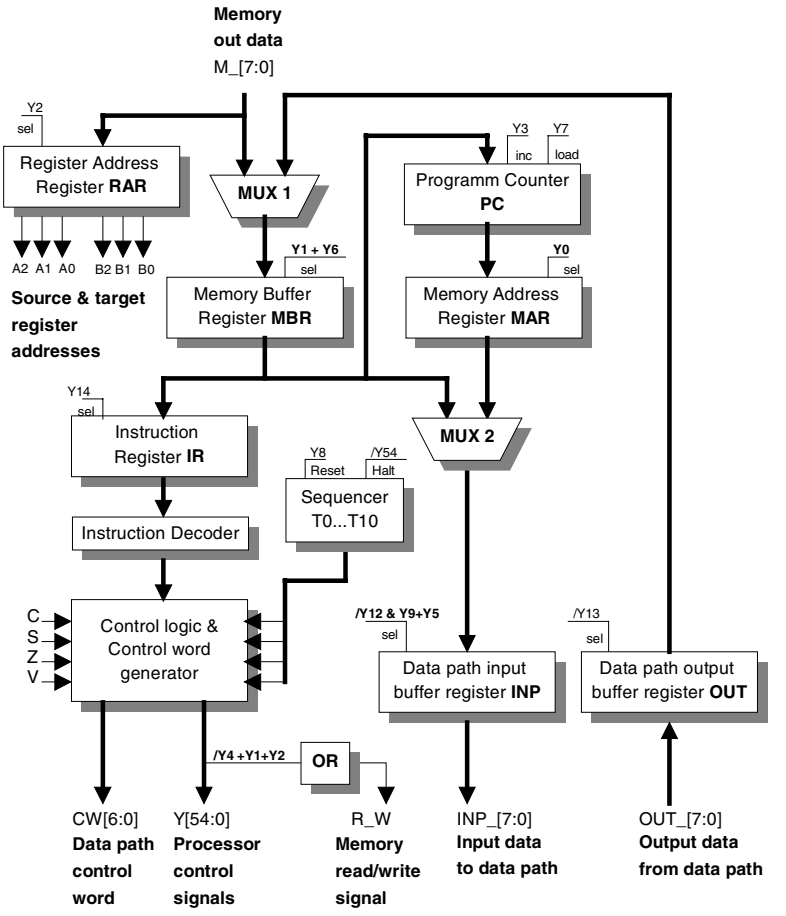
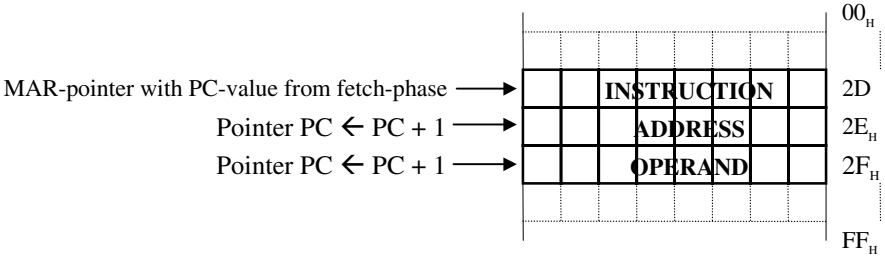


Fig. 59. Control Path of t4008

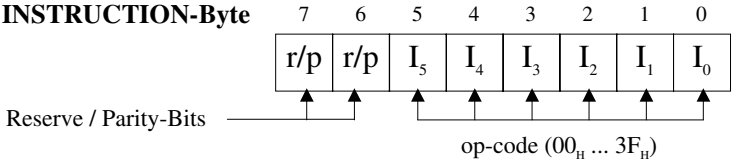
With the currently used 8-bit MAR a space of 256 memory cells is addressable. The memory width is 8-bit. For every instruction-′block′ (*opcode+operand*), 3 bytes are provided.

The following figure illustrates the used instruction format:

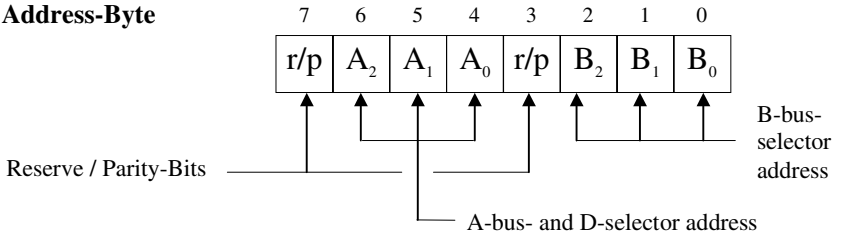


Format of instruction-block bytes:

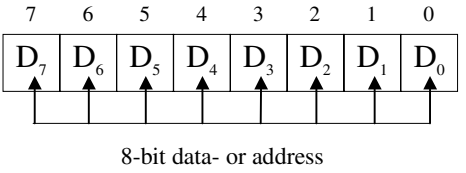
1. INSTRUCTION-Byte



2. Address-Byte



3. Operand



Example 1: Two stored 8-bit numbers should be added . The result should be written back to the memory. (start address = $2D_H$):

```
JMP g 2d      # rescue old address in register 7 (=g) and jump to 2dH
MVX a 2f      # write data at 2fH to register 1
MVX b 32      # write data at 32H to register 2
ADD a b       # add a and b, write result to a
MVM a 14      # write a to memory cell 14H
RET g         # return to address[g]
```

The binary code in the memory is as follows:

← Start address =00_H

JMP	x	x	1	1	0	0	0	0
g	x	1	1	1	x	0	0	0
2d	0	0	1	0	1	1	0	1
...	...	...	...	...	...	...	...	...

← Jump address =$2d_H$

MVX	x	x	0	0	0	0	1	0
a	x	0	0	1	x	0	0	0
2f	0	1	0	1	1	1	0	1

MVX	x	x	0	0	0	0	1	0
b	x	0	1	0	x	0	0	0
32	1	0	1	0	0	0	1	0
ADD	x	x	0	0	1	0	0	0
a b	x	0	0	1	x	0	1	0
	x	x	x	X	x	x	x	x
MVM	x	x	0	0	0	0	1	1
a	x	0	0	0	x	0	0	1
14	0	0	0	1	0	1	0	0
RET	x	x	1	1	0	1	1	1
g	x	0	0	0	x	1	1	1
	x	x	x	X	x	x	x	x
...	...	...						
← Result at 14_H								
result	1	1	1	1	1	1	1	1

The following table contains the instruction set of processor t4008:

Table 22. t4008 Instruction Set

	Mnemonic	Format	Register transfer	$I_5 \dots I_0$	A_2	A_1	A_0	B_2	B_1	B_0
00	MVIN x	x = a,b,c,d,e,f,g	Rx ← Input-Port	000000	A	A	A	x	x	x
01	MOV x y	x,y=a,b,c,d,e,f,g	Rx ← Ry	000001	A	A	A	B	B	B
02	MVX x m	x=a,b,..,g; m=XX_H	Rx ← M	000010	A	A	A	0	0	0
03	MVM x m	x=a,b,..,g; m=XX_H	M ← Rx	000011	0	0	0	B	B	B
04	INCX x	x=a,b,..,g	Rx ← Rx+1	000100	A	A	A	x	x	x
05	INCM m	m=XX_H	M ← M+1	000101	0	0	0	x	x	x
06	DECX x	x=a,b,..,g	Rx ← Rx-1	000110	A	A	A	x	x	x
07	DECM m	m=XX_H	M ← M-1	000111	0	0	0	x	x	x
08	ADD x y	x,y=a,b,c,d,e,f,g	Rx ← Rx+Ry	001000	A	A	A	B	B	B
09	ADX x m	x=a,b,..,g; m=XX_H	Rx ← Rx+M	001001	A	A	A	0	0	0
0A	ADM x m	x=a,b,..,g; m=XX_H	M ← M+Rx	001010	0	0	0	B	B	B
0B	ADC x y	x,y=a,b,c,d,e,f,g	Rx ← Rx+Ry+C	001011	A	A	A	B	B	B
0C	ACX x m	x=a,b,..,g; m=XX_H	Rx ← Rx+M+C	001100	A	A	A	0	0	0
0D	ACM x m	x=a,b,..,g; m=XX_H	M ← M+Rx+C	001101	0	0	0	B	B	B
0E	SUB x y	x,y=a,b,c,d,e,f,g	Rx ← Rx-Ry	001110	A	A	A	B	B	B
0F	SUX x m	x=a,b,..,g; m=XX_H	Rx ← Rx-M	001111	A	A	A	0	0	0
10	SUM x m	x=a,b,..,g; m=XX_H	M ← M-Rx	010000	0	0	0	B	B	B
11	SBB x y	x,y=a,b,c,d,e,f,g	Rx ← Rx-Ry-1	010001	A	A	A	B	B	B
12	SBX x m	x=a,b,..,g; m=XX_H	Rx ← Rx-M-1	010010	A	A	A	0	0	0
13	SBM x m	x=a,b,..,g; m=XX_H	M ← M-Rx-1	010011	0	0	0	B	B	B
14	not used			010100						
15	not used			010101						
16	AND x y	x,y=a,b,c,d,e,f,g	Rx ← Rx ∧ Ry	010110	A	A	A	B	B	B
17	ANX x m	x=a,b,..,g; m=XX_H	Rx ← Rx ∧ M	010111	A	A	A	0	0	0
18	ANM x m	x=a,b,..,g; m=XX_H	M ← M ∧ Rx	011000	0	0	0	B	B	B
19	OR x y	x,y=a,b,c,d,e,f,g	Rx ← Rx ∨ Ry	011001	A	A	A	B	B	B
1A	ORX x m	x=a,b,..,g; m=XX_H	Rx ← Rx ∨ M	011010	A	A	A	0	0	0
1B	ORM x m	x=a,b,..,g; m=XX_H	M ← M ∨ Rx	011011	0	0	0	B	B	B
1C	XOR x y	x,y=a,b,c,d,e,f,g	Rx ← Rx ⊕ Ry	011100	A	A	A	B	B	B

Table 22. (continued)

1D	XOX x m	x=a,b,..,g; m=XX_H	Rx ← Rx ⊕ M	011101	A	A	A	0	0	0
1E	XOM x m	x=a,b,..,g; m=XX_H	M ← M ⊕ Rx	011110	0	0	0	B	B	B
1F	CMP x y	x,y=a,b,c,d,e,f,g	Z ← 1 if (x=y), C ← 1 if (x>y)	011111	A	A	A	B	B	B
20	CPX x m	x=a,b,..,g; m=XX_H	Z ← 0 if (x≠y), C ← 0 if (x<y)	100000	A	A	A	0	0	0
21	CPM x m	x=a,b,..,g; m=XX_H	dito	100001	0	0	0	B	B	B
22	SHRX x	x=a,b,..,g	shift-right Rx, R_7 ← 0	100010	A	A	A	x	x	x
23	SHRM m	m=XX_H	shift-right Mx, M_7 ← 0	100011	0	0	0	x	x	x
24	SHLX x	x=a,b,..,g	shift-left Rx, R_0 ← 0	100100	A	A	A	x	x	x
25	SHLM m	m=XX_H	shift-left M, M_0 ← 0	100101	0	0	0	x	x	x
26	CRCX x	x=a,b,..,g	circulate right with carry	100110	A	A	A	x	x	x
27	CRCM m	m=XX_H	circulate right with carry	100111	0	0	0	x	x	x
28	CLCX x	x=a,b,..,g	circulate left with carry	101000	A	A	A	x	x	x
29	CLCM m	m=XX_H	circulate left with carry	101001	0	0	0	x	x	x
2A	CMLX x	x=a,b,..,g	Rx ← Rx'	101010	A	A	A	x	x	x
2B	CMLM m	m=XX_H	M ← M'	101011	0	0	0	x	x	x
2C	SETC		C ← 1	101100	0	0	0	x	x	x
2D	CLR x	x=a,b,..,g	Rx ← 0	101101	A	A	A	x	x	x
2E	CLM m			101110	0	0	0	x	x	x
2F	CLC		C ← 0	101111	0	0	0	x	x	x
30	JMP x m	x=a,b,..,g; m=XX_H	Rx ← PC, PC ← new adr[m]	110000	A	A	A	0	0	0
31	JNZ x m	x=a,b,..,g; m=XX_H	dito if Z=0	110001	A	A	A	0	0	0
32	JZ x m	x=a,b,..,g; m=XX_H	dito if Z=1	110010	A	A	A	0	0	0
33	JNC x m	x=a,b,..,g; m=XX_H	dito if C=0	110011	A	A	A	0	0	0
34	JC x m	x=a,b,..,g; m=XX_H	dito if C=1	110100	A	A	A	0	0	0
35	JP x m	x=a,b,..,g; m=XX_H	dito if S=0	110101	A	A	A	0	0	0
36	JM x m	x=a,b,..,g; m=XX_H	dito if S=1	110110	A	A	A	0	0	0
37	RET x	x=a,b,..,g	PC ← adr[stored in Rx]	110111	0	0	0	B	B	B
38	RNZ x	x=a,b,..,g	PC ← adr[stored in Rx] if Z=0	111000	0	0	0	B	B	B
39	RZ x	x=a,b,..,g	PC ← adr[stored in Rx] if Z=1	111001	0	0	0	B	B	B
3A	RNC x	x=a,b,..,g	PC ← adr[stored in Rx] C=0	111010	0	0	0	B	B	B
3B	RC x	x=a,b,..,g	PC ← adr[stored in Rx] C=1	111011	0	0	0	B	B	B
3C	RP x	x=a,b,..,g	PC ← adr[stored in Rx] if S=0	111100	0	0	0	B	B	B
3D	RM x	x=a,b,..,g	PC ← adr[stored in Rx] S=1	111101	0	0	0	B	B	B
3E	not used			111110						
3F	NOP		no operation	111111	x	x	x	x	x	x

A.2 Microprocessors t5008/16/32x

Processors t5008x, t5016x and t5032x represent a common instruction set similar to Intel's 8085/86, Motorola's 68000 and so one. All instruction sets base on a dynamic instruction length. The last number in the name represents the according bit range for operands. Processors named with a letter **m**, contain a fast multiplier. For instance, t5016m has a fast 8x8-bit multiplier. Letter **p** represents a pipelined instruction execution. The basic architecture is similar for all t50xx-designs. For explanation of this processor type, the t5016m is considered as a representative design.

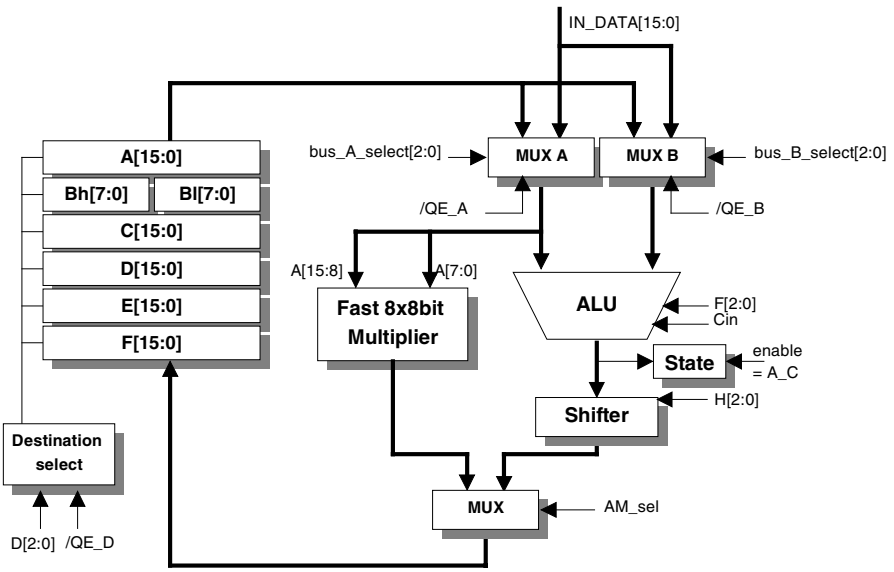


Fig. 60. t5016m Data-Path Block Diagram

The top-level contains also the data- and control-path as well as a RAM macro. The data-path is similar to the previous t4008-processor. The B-register is partitioned into two registers half of the bit length of 16-bit in order to support 8-bit operations or to swap high and low byte.

Table 23. Bus Multiplexer Control

/QE_A/B	bus_A/B_ select[2:0]	MUXA/B-out
0	XXX	0
1	000	IN[15:0]
1	001	A[15:0]
1	010	B[15:0]
1	011	B[15:0]
1	100	C[15:0]
1	101	D[15:0]
1	110	E[15:0]
1	111	F[15:0]

Table 24. Destination Select Signals

/QE_D	D[2:0]	write register
1	XXX	0
0	000	0
0	001	register A
0	010	register BH
0	011	register BL
0	100	register C
0	101	register D
0	110	register E
0	111	register F

Operands can be pass the integer ALU or the (fast) 8x8-bit multiplier. The multiplier is able to multiply the high byte with the low byte of a register content. Next tables define the control of multiplexers `MUX A`, `MUX B` and destination selector for the register-file. The control-path structure can be derived from Fig. 15. In contrast to t4008, it contains a stack-pointer SP to rescue processor state, in case of a branching.

Instructions are implicit and support immediate data, absolute and indirect addressing. The following figure outlines the implemented instruction format. The format includes one byte length for implicit instruction, two bytes for instructions with immediate 8-bit data and three bytes for instructions with 16-bit data or address.

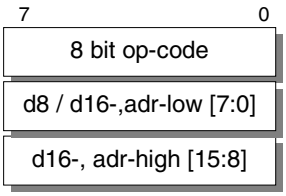


Fig. 61. t5016-Instruction Format

The memory is organized as follow: In the current implementation, a 64k area of an 8-bit RAM can be addressed. The Stack memory – addressed with stack-pointer SP – is included in a reserved area.

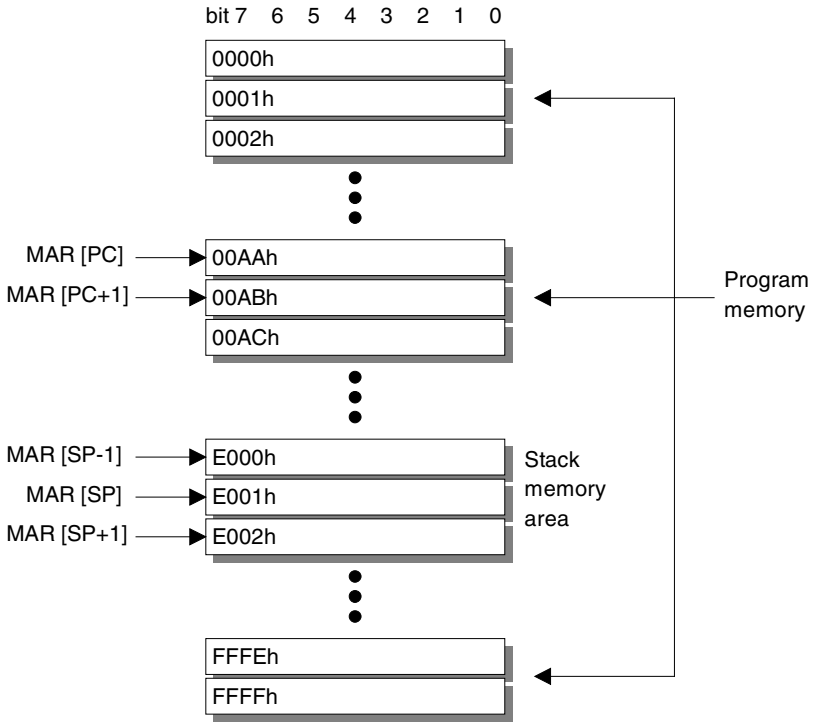


Fig. 62. t5016-Memory Organization

Following examples outline the basics of micro-operations and according register transfers. First, the fetch-phase is illustrated:

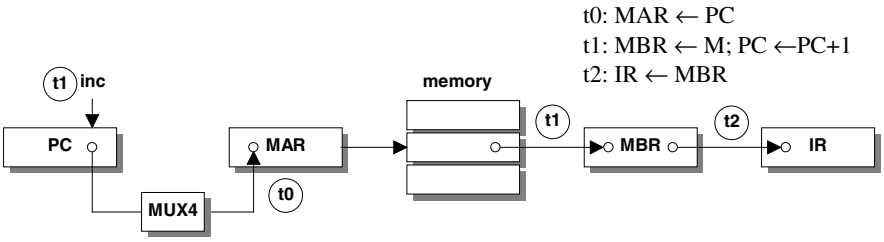


Fig. 63. t5016-Instruction Fetch Phase

The instruction **MOV B,A** needs the following data-path control signals:

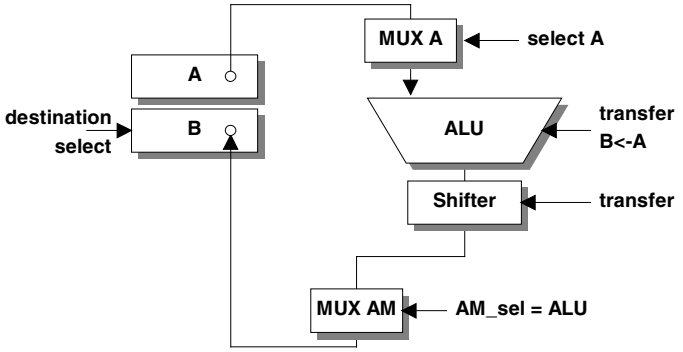


Fig. 64. t5016-Control Signals for the Execution of MOV B,A

The following table lists the complete instruction set of the t5016m microprocessor.

Table 25. t5016m – Instruction Set

Op-code			Mnemonic	Function
Hex	Binary	Instr. variable		
41	0100 0001	q65	**MVM A**	move register A to the memory cell with address in BH,BL
42	0100 0010	q66	**MVM BH**	move register BH to the memory cell with address in BH,BL
43	0100 0011	q67	**MVM BL**	move register BL to the memory cell with address in BH,BL
44	0100 0100	q68	**MVM C**	move register C to the memory cell with address in BH,BL
45	0100 0101	q69	**MVM D**	move register D to the memory cell with address in BH,BL
46	0100 0110	q70	**MVM E**	move register E to the memory cell with address in BH,BL
47	0100 0111	q71	**MVM F**	move register F to the memory cell with address in BH,BL
48	0100 1000	q72	**MVX A**	load mem-cell M (8-bit) (address BH,BL) to the register A
49	0100 1001	q73	**MOV A,A**	register transfer from A to A
4A	0100 1010	q74	**MOV A,B**	register transfer from B to A
4C	0100 1100	q76	**MOV A,C**	register transfer from C to A
4D	0100 1101	q77	**MOV A,D**	register transfer from D to A
4E	0100 1110	q78	**MOV A,E**	register transfer from E to A
4F	0100 1111	q79	**MOV A,F**	register transfer from F to A
50	0101 0000	q80	**MVX BH**	load mem-cell M (8-bit) (address BH,BL) to the register BH

Table 25. (continued)

51	0101 0001	q81	**MOV BH,A**	transf. register A high-byte (A[15:8]) to BH
54	0101 0100	q84	**MOV BH,C**	transf. register C high-byte (C[15:8]) to BH
55	0101 0101	q85	**MOV BH,D**	transf. register D high-byte (DA[15:8]) to BH
56	0101 0110	q86	**MOV BH,E**	transf. register E high-byte (E[15:8]) to BH
57	0101 0111	q87	**MOV BH,F**	transf. register F high-byte (F[15:8]) to BH
58	0101 1000	q88	**MVX BL**	load mem-cell M (8-bit) (address BH,BL) to the register BL
59	0101 1001	q89	**MOV BL,A**	transfer of the register A low-byte (A[7:0]) to BL
5C	0101 1100	q92	**MOV BL,C**	transfer of the register C low-byte (C[7:0]) to BL
5D	0101 1101	q93	**MOV BL,D**	transfer of the register D low-byte (D[7:0]) to BL
5E	0101 1110	q94	**MOV BL,E**	transfer of the register E low-byte (E[7:0]) to BL
5F	0101 1111	q95	**MOV BL,F**	transfer of the register F low-byte (F[7:0]) to BL
60	0110 0000	q96	**MVX C**	load mem-cell M (8-bit) (address BH,BL) to the register C
61	0110 0001	q97	**MOV C,A**	register transfer from A to C
62	0110 0010	q98	**MOV C,B**	register transfer from B to C
64	0110 0100	q100	**MOV C,C**	register transfer from C to C
65	0110 0101	q101	**MOV C,D**	register transfer from D to C
66	0110 0110	q102	**MOV C,E**	register transfer from E to C
67	0110 0111	q103	**MOV C,F**	register transfer from F to C
68	0110 1000	q104	**MVX D**	load mem-cell M (8-bit) (address BH,BL) to the register D
69	0110 1001	q105	**MOV D,A**	register transfer from A to D
6A	0110 1010	q106	**MOV D,B**	register transfer from B to D
6C	0110 1100	q108	**MOV D,C**	register transfer from C to D
6D	0110 1101	q109	**MOV D,D**	register transfer from D to D
6E	0110 1110	q110	**MOV D,E**	register transfer from E to D
6F	0110 1111	q111	**MOV D,F**	register transfer from F to D
70	0111 0000	q112	**MVX E**	load mem-cell M (8-bit) (address BH,BL) to the register E
71	0111 0001	q113	**MOV E,A**	register transfer from A to E
72	0111 0010	q114	**MOV E,B**	register transfer from B to E
74	0111 0100	q116	**MOV E,C**	register transfer from C to E
75	0111 0101	q117	**MOV E,D**	register transfer from D to E
76	0111 0110	q118	**MOV E,E**	register transfer from E to E
77	0111 0111	q119	**MOV E,F**	register transfer from F to E
78	0111 1000	q120	**MVX F**	load mem-cell M (8-bit) (address BH,BL) to the register F
79	0111 1001	q121	**MOV F,A**	register transfer from A to F
7A	0111 1010	q122	**MOV F,B**	register transfer from B to F
7C	0111 1100	q124	**MOV F,C**	register transfer from C to F
7D	0111 1101	q125	**MOV F,D**	Register transfer from D to F
7E	0111 1110	q126	**MOV F,E**	Register transfer from E to F

Table 25. (continued)

7F	0111 1111	q127	**MOV F,F**	register transfer from F to F
10	0001 0000	q16	**MVI BH,d8**	2 Byte – move directly a 8 bit-data (d8) to BH
18	0001 1000	q24	**MVI BL,d8**	2 Byte – move directly a 8 bit-data (d8) to BL
08	0000 1000	q8	**LXI A,d16**	3 Byte – move directly a 16 bit-data (d16) to register A
20	0010 0000	q32	**LXI C,d16**	3 Byte – move directly a 16 bit-data (d16) to register C
28	0010 1000	q40	**LXI D,d16**	3 Byte – move directly a 16 bit-data (d16) to register D
30	0011 0000	q48	**LXI E,d16**	3 Byte – move directly a 16 bit-data (d16) to register E
38	0011 1000	q56	**LXI F,d16**	3 Byte – move directly a 16 bit-data (d16) to register F
12	0001 0010	q18	**LDBH adr**	3 Byte – load data with address (adr) directly to BH
1A	0001 1010	q26	**LDBL adr**	3 Byte – load data with address (adr) directly to BL
01	0000 0001	q1	**STA adr**	3 Byte – store reg. A in the memory cell M with address (adr)
02	0000 0010	q2	**STBH adr**	3 Byte – store reg. BH in the memory cell with address (adr)
03	0000 0011	q3	**STBL adr**	3 Byte – store reg. BL in the memory cell with address (adr)
04	0000 0100	q4	**STC adr**	3 Byte – store reg. C in the memory cell M with address (adr)
05	0000 0101	q5	**STD adr**	3 Byte – store reg. D in the memory cell M with address (adr)
06	0000 0110	q6	**STE adr**	3 Byte – store reg. E in the memory cell M with address (adr)
07	0000 0111	q7	**STF adr**	3 Byte – store reg. F in the memory cell M with address (adr)
19	0001 1001	q25	**LDAX A**	load memory cell with address in A directly to register A
21	0010 0001	q33	**LDAX B**	load memory cell with address in B directly to register A
22	0010 0010	q34	**LDAX C**	load memory cell with address in C directly to register A
23	0010 0011	q35	**LDAX D**	load memory cell with address in D directly to register A
26	0010 0110	q38	**LDAX E**	load memory cell with address in E directly to register A
27	0010 0111	q39	**LDAX F**	load memory cell with address in F directly to register A
D1	1101 0001	q209	**PUSH A**	transfer A-content to the stack (SP-1=A-low; SP-2=A-high)
D5	1101 0101	q213	**PUSH B**	transfer B-content to the stack (SP-1=B-low; SP-2=B-high)
CF	1100 1111	q207	**PUSH C**	transfer C-content to the stack (SP-1=C-low; SP-2=C-high)
D3	1101 0011	q211	**PUSH D**	transfer D-content to the stack (SP-1=D-low; SP-2=D-high)
D9	1101 1001	q217	**PUSH E**	transfer E-content to the stack (SP-1=E-low; SP-2=E-high)
DF	1101 1111	q223	**PUSH F**	transfer F-content to the stack (SP-1=F-low; SP-2=F-high)

Table 25. (continued)

31	0011 0001	q49	**POP A**	transfer stack-content to A (SP=high-byte; SP+1=low-byte)
32	0011 0010	q50	**POP B**	transfer stack-content to B (SP=high-byte; SP+1=low-byte)
33	0011 0011	q51	**POP C**	transfer stack-content to C (SP=high-byte; SP+1=low-byte)
36	0011 0110	q54	**POP D**	transfer stack-content to D (SP=high-byte; SP+1=low-byte)
39	0011 1001	q57	**POP E**	transfer stack-content to E (SP=high-byte; SP+1=low-byte)
3F	0011 1111	q63	**POP F**	transfer stack-content to F (SP=high-byte; SP+1=low-byte)
DB	1101 1011	q219	**INP A**	write input-port data into register A
09	0000 1001	q9	**INP B**	write input-port data into register B
0A	0000 1010	q10	**INP C**	write input-port data into register C
0B	0000 1011	q11	**INP D**	write input-port data into register D
0E	0000 1110	q14	**INP E**	write input-port data into register E
0F	0000 1111	q15	**INP F**	write input-port data into register F
DD	1101 1101	q221	**OUT A**	put data from A register to the output-port
13	0001 0011	q19	**OUT B**	put data from B register to the output-port
14	0001 0100	q20	**OUT C**	put data from C register to the output-port
15	0001 0101	q21	**OUT D**	put data from D register to the output-port
16	0001 0110	q22	**OUT E**	put data from E register to the output-port
17	0001 0111	q23	**OUT F**	put data from F register to the output-port
80	1000 0000	q128	**ADD M**	add A with M [BH,BL] and transfer sum to A
81	1000 0001	q129	**ADD A**	add A with A and transfer the sum to A
82	1000 0010	q130	**ADD B**	add B with A and transfer the sum to A
84	1000 0100	q132	**ADD C**	add C with A and transfer the sum to A
85	1000 0101	q133	**ADD D**	add D with A and transfer the sum to A
86	1000 0110	q134	**ADD E**	add E with A and transfer the sum to A
87	1000 0111	q135	**ADD F**	add F with A and transfer the sum to A
83	1000 0011	q131	**ADI d16**	3 Byte – add a 16-bit–data (d16) to A and transfer the sum to register A
88	1000 1000	Q136	**ADC M**	add A with M[BH,BL] plus carry and transfer the sum to register A
89	1000 1001	Q137	**ADC A**	add A with A plus carry and transfer the sum to register A
8A	1000 1010	Q138	**ADC B**	add B with A plus carry and transfer the sum to register A
8C	1000 1100	q140	**ADC C**	add C with A plus carry and transfer the sum to register A
8D	1000 1101	q141	**ADC D**	add D with A plus carry and transfer the sum to register A
8E	1000 1110	q142	**ADC E**	add E with A plus carry and transfer the sum to register A
8F	1000 1111	q143	**ADC F**	Add F with A plus carry and transfer the sum to register A
DE	1101 1110	q222	**ACI d16**	3 Byte – add data (d16) to A plus carry; transfer the sum to register A
90	1001 0000	q144	**SUB M**	subtract A–M[BH,BL] and transfer the difference to register A

Table 25. (continued)

91	1001 0001	q145	**SUB A**	subtract A–A and transfer the difference to A
92	1001 0010	q146	**SUB B**	subtract A–B and transfer the difference to A
94	1001 0100	q148	**SUB C**	subtract A–C and transfer the difference to A
95	1001 0101	q149	**SUB D**	subtract A–D and transfer the difference to A
96	1001 0110	q150	**SUB E**	subtract A–E and transfer the difference to A
97	1001 0111	q151	**SUB F**	subtract A–F and transfer the difference to A
D6	1101 0110	q214	**SUI d16**	3 Byte – subtract A – d16 and transfer the difference to register A
98	1001 1000	q152	**SBB M**	subtr. A – M[BH,BL] – borrow-bit; transfer the difference to register A
99	1001 1001	q153	**SBB A**	subtract A – A – borrow-bit; transfer the the difference to register A
9A	1001 1010	q154	**SBB B**	subtract A – B – borrow-bit; transfer the difference to register A
9C	1001 1100	q156	**SBB C**	subtract A – C – borrow-bit; transfer the difference to register A
9D	1001 1101	q157	**SBB D**	subtract A – D – borrow-bit; transfer the difference to register A
9E	1001 1110	q158	**SBB E**	subtract A – E – borrow-bit; transfer the difference to register A
9F	1001 1111	q159	**SBB F**	subtract A – F – borrow-bit; transfer the difference to register A
F1	1111 0001	q241	**SBI d16**	3 Byte – Subtr. A – d16 – borrow-bit; transfer the difference to register A
11	0001 0001	q17	**INR M**	increment memory cell M with address in register [BH, BL]
0C	0000 1100	q12	**INR A**	increment register A
24	0010 0100	q36	**INR C**	increment register C
2C	0010 1100	q44	**INR D**	increment register D
34	0011 0100	q52	**INR E**	increment register E
3C	0011 1100	q60	**INR F**	increment register F
29	0010 1001	q41	**DCR M**	decrement memory cell M with address in [BH, BL]
0D	0000 1101	q13	**DCR A**	decrement memory cell M with address A
25	0010 0101	q37	**DCR C**	decrement memory cell M with address C
2D	0010 1101	q45	**DCR D**	decrement memory cell M with address D
35	0011 0101	q53	**DCR E**	decrement memory cell M with address E
3D	0011 1101	q61	**DCR F**	decrement memory cell M with address F
E0	1110 0000	q224	**MUL M**	load A with the product high-byte · low-byte of M[BH, BL]
E1	1110 0001	q225	**MUL A**	load A with the product high-byte · low-byte of A
E2	1110 0010	q226	**MUL B**	load A with the product high-byte · low-byte of B
E4	1110 0100	q228	**MUL C**	load A with the product high-byte · low-byte of C
E5	1110 0101	q229	**MUL D**	load A with the product high-byte · low-byte of D
E6	1110 0110	q230	**MUL E**	load A with the product high-byte · low-byte of E

Table 25. (continued)

E7	1110 0111	q231	**MUL F**	load A with the product high-byte · low-byte of F
E8	1110 1000	q232	**MIBH d8**	2 Byte – load A with the product (BH · d8); (BL will be lost)
E9	1110 1001	q233	**MIBL d8**	2 Byte – load A with the product (BL · d8); (BH will be lost)
EA	1110 1010	q234	**MUX**	3 Byte – Multiplication d16[15:8] · d16[7:0]; transfer to A
A0	1010 0000	q160	**ANA M**	load A with result from high- AND low-Byte of M[BH, BL]
A1	1010 0001	q161	**ANA A**	load A with the result from A AND A
A2	1010 0010	q162	**ANA B**	load A with the result from A AND B
A4	1010 0100	q164	**ANA C**	load A with the result from A AND C
A5	1010 0101	q165	**ANA D**	load A with the result from A AND D
A6	1010 0110	q166	**ANA E**	load A with the result from A AND E
A7	1010 0111	q167	**ANA F**	load A with the result from A AND F
C7	1100 0111	q199	**ANI d16**	3 Byte - load A with result from d16 AND A
B0	1011 0000	q176	**ORA M**	load A with the result from high- OR low-byte of M[BH, BL]
B1	1011 0001	q177	**ORA A**	load A with the result from A OR A
B2	1011 0010	q178	**ORA B**	load A with the result from A OR B
B4	1011 0100	q180	**ORA C**	load A with the result from A OR C
B5	1011 0101	q181	**ORA D**	load A with the result from A OR D
B6	1011 0110	q182	**ORA E**	load A with the result from A OR E
B7	1011 0111	q183	**ORA F**	load A with the result from A OR F
F6	1111 0110	q246	**ORI d16**	3 Byte – load A with result from d16 OR A
A8	1010 1000	q168	**XRA M**	load A with the result from high- XOR low-byte of M[BH, BL]
A9	1010 1001	q169	**XRA A**	load A with the result from A XOR A
AA	1010 1010	q170	**XRA B**	load A with the result from A XOR B
AC	1010 1100	q172	**XRA C**	load A with the result from A XOR C
AD	1010 1101	q173	**XRA D**	load A with the result from A XOR D
AE	1010 1110	q174	**XRA E**	load A with the result from A XOR E
AF	1010 1111	q175	**XRA F**	load A with the result from A XOR F
3E	0011 1110	q62	**XRI d16**	3 Byte – load A with result from d16 XOR A
3A	0011 1010	q58	**RLC**	rotate the accumulator A left circular
3B	0011 1011	q59	**RRC**	rotate the accumulator A right circular
2F	0010 1111	q47	**CMA**	load A with the complement of A
37	0011 0111	q55	**SEC**	set the carry-Flag
B8	1011 1000	q184	**CPI d16**	3 Byte – compare A with immediate data d16
B9	1011 1001	q185	**CMP M**	compare A with memory the cell M[BH, BL] and set flags
BA	1011 1010	q186	**CMP A**	compare A with A and set flags
BB	1011 1011	q187	**CMP B**	compare A with B and set flags
BC	1011 1100	q188	**CMP C**	compare A with C and set flags
BD	1011 1101	q189	**CMP D**	compare A with D and set flags
BE	1011 1110	q190	**CMP E**	compare A with E and set flags

Table 25. (continued)

BF	1011 1111	q191	**CMP F**	compare A with F and set flags
C3	1100 0011	q195	**JMP adr**	3 Byte – jump unconditionally to the memory address (adr)
C2	1100 0010	q194	**JNZ adr**	3 Byte – jump to adress if Zero-flag = 0
CA	1100 1010	q202	**JZ adr**	3 Byte – jump to adress if Zero-flag = 1
D2	1101 0010	q210	**JNC adr**	3 Byte – jump to adress if Carry-flag = 0
DA	1101 1010	q218	**JC adr**	3 Byte – jump to adress if Carry-flag = 1
F2	1111 0010	q242	**JP adr**	3 Byte – jump to adress if Sign-flag = 0
FA	1111 1010	q250	**JM adr**	3 Byte – jump to adress if Sign-flag = 1
CD	1100 1101	q205	**CALL adr**	3 Byte – routine call at adr, rescue actual address to the Stack
C4	1100 0100	q196	**CNZ adr**	3 Byte – routine call if Z=0, rescue actual address to the Stack
CC	1100 1100	q204	**CZ adr**	3 Byte – routine call if Z=1, rescue actual address to the Stack
D4	1101 0100	q212	**CNC adr**	3 Byte – routine call if C=0, rescue actual address to the Stack
DC	1101 1100	q220	**CC adr**	3 Byte – routine call if C=1, rescue actual address to the Stack
F4	1111 0100	q244	**CP adr**	3 Byte – routine call if S=0, rescue actual address to the Stack
FC	1111 1100	q252	**CM adr**	3 Byte – routine call if S=1, rescue actual address to the Stack
C9	1100 1001	q201	**RET**	3 Byte – return to the rescued address stored to the Stack
C0	1100 0000	q192	**RNZ**	3 Byte – return to the rescued address stored to the Stack if Z=0
C8	1100 1000	q200	**RZ**	3 Byte – return to the rescued address stored to the Stack if Z=1
D0	1101 0000	q208	**RNC**	3 Byte – return to the rescued addr. stored to the Stack if C=0
D8	1101 1000	q216	**RC**	3 Byte – return to the rescued addr. stored to the Stack if C=1
F0	1111 0000	q240	**RP**	3 Byte – return to the rescued addr. stored to the Stack if S=0
F8	1111 1000	q248	**RM**	3 Byte – return to the rescued addr. stored to the Stack if S=1
00	0000 0000	q0	**NOP**	no operation – program pause
1B	0001 1011	q27	**LXSP d16**	load stack pointer with immediate–data (d16)
1C	0001 1100	q28	**LDSP**	load stack pointer with data from register B

A.3 Digital Signal Processors uDSP32a/b

Processors uDSP32a and uDSP32b are universal digital signal processors with a data processing bit-width of 32-bit. The structure of the version ′a′ is similar to the architecture of the t5032 to process 32-bit integer data. The 32-bit instruction format (bits `[31:24]` = op-code; bits `[23:0]` = data memory address) is oriented to the Harvard-principle: The memory space for instructions and data are separately addressable. Therefor, the control-path consists of two memory address registers. Special instructions are implemented in order to load or store double data or six data words in

order to support faster data processing. DSP version uDSP32b consists of single precision floating point addition, subtraction, and multiplication.

Table 26. Micro -Operations of the Fetch-Phase

Mnemonic	q_xt_x	Control Signals	Micro-operations	Function
Fetch	q_xt_0 q_xt_1 q_xt_2	y_0 y_1, y_2, y_3 y_4, y_{10}	IMAR ← PC IMBR ← IM, PC ← PC + 1, DMAR ← IM IR ← IMBR, READBUFFER ← INP	instruction and operand fetch

Table 27. Single-Data Load/Store Instruction

Mnemonic	q_xt_x	Control signal	Micro-Operation	Function
LD A,addr	q_1t_3 q_1t_4	y_{13} y_9	A ← READBUFFER T ← 0	load data at address to register A
LD B,addr	q_2t_3 q_2t_4	y_{14} y_9	B ← READBUFFER T ← 0	load data at address to register B
LD C,addr	q_3t_3 q_3t_4	y_{15} y_9	C ← READBUFFER T ← 0	load data at address to register C
LD D,addr	q_4t_3 q_4t_4	y_{16} y_9	D ← READBUFFER T ← 0	load data at address to register D
LD E,addr	q_5t_3 q_5t_4	y_{17} y_9	E ← READBUFFER T ← 0	load data at address to register E
LD F,addr	q_6t_3	y_{18}	F ← READBUFFER	load data at address to register F
ST addr,A	q_7t_3 q_7t_4 q_7t_5	y_{11}, y_{19} y_{12} y_9	WRITEBUFFER ← A OUT ← WRITEBUFFER T ← 0	store register A to memory (address)
ST addr,B	q_8t_3 q_8t_4 q_8t_4	y_{11}, y_{20} y_{12} y_9	WRITEBUFFER ← B OUT ← WRITEBUFFER T ← 0	store register B to memory (address)
ST addr,C	q_9t_3 q_9t_4 q_9t_4	y_{11}, y_{21} y_{12} y_9	WRITEBUFFER ← C OUT ← WRITEBUFFER T ← 0	store register C to memory (address)
ST addr,D	$q_{10}t_3$ $q_{10}t_4$ $q_{10}t_4$	y_{11}, y_{22} y_{12} y_9	WRITEBUFFER ← D OUT ← WRITEBUFFER T ← 0	store register D to memory (address)
ST addr,E	$q_{11}t_3$ $q_{11}t_4$ $q_{11}t_4$	y_{11}, y_{23} y_{12} y_9	WRITEBUFFER ← E OUT ← WRITEBUFFER T ← 0	store register E to memory (address)
ST addr,F	$q_{12}t_3$ $q_{12}t_4$ $q_{12}t_4$	y_{11}, y_{24} y_{12} y_9	WRITEBUFFER ← F OUT ← WRITEBUFFER T ← 0	store register F to memory (address)

Table 28. Double-Data Load/Store Instruction

Mnemonic	q_xt_x	Control Signal	Micro-Operation	Function
LDB AB,addr	$q_{13}t_3$	y_{13}, y_6	A ← READBUFFER, DMAR ← DMAR + 1	load two data words to register A and B
	$q_{13}t_4$	y_{10}	READBUFFER ← INP	
	$q_{13}t_5$	y_{14}	B ← READBUFFER	
	$q_{13}t_6$	y_9	T ← 0	
LDB CD,addr	$q_{14}t_3$	y_{15}, y_6	C ← READBUFFER, DMAR ← DMAR + 1	load two data words to register C and D
	$q_{14}t_4$	y_{10}	READBUFFER ← INP	
	$q_{14}t_5$	y_{16}	D ← READBUFFER	
	$q_{14}t_6$	y_9	T ← 0	
LDB EF,addr	$q_{15}t_3$	y_{17}, y_6	E ← READBUFFER, DMAR ← DMAR + 1	load two data words to register E and F
STB addr,AB	$q_{16}t_3$	y_{11}, y_{19}	WRITEBUFFER ← A	store register A and B to the memory (address)
	$q_{16}t_4$	y_{12}	OUT ← WRITEBUFFER	
	$q_{16}t_5$	y_{11}, y_{20}, y_6	WRITEBUFFER ← B, DMAR ← DMAR + 1	
	$q_{16}t_6$	y_{12}	OUT ← WRITEBUFFER	
	$q_{16}t_7$	y_9	T ← 0	
STB addr,CD	$q_{17}t_3$	y_{11}, y_{21}	WRITEBUFFER ← C	store register C and D to the memory (address)
	$q_{17}t_4$	y_{12}	OUT ← WRITEBUFFER	
	$q_{17}t_5$	y_{11}, y_{22}, y_6	WRITEBUFFER ← D, DMAR ← DMAR + 1	
	$q_{17}t_6$	y_{12}	OUT ← WRITEBUFFER	
	$q_{17}t_7$	y_9	T ← 0	
STB addr,EF	$q_{18}t_3$	y_{11}, y_{23}	WRITEBUFFER ← E	store register E and F to the memory (address)
	$q_{18}t_4$	y_{12}	OUT ← WRITEBUFFER	
	$q_{18}t_5$	y_{11}, y_{24}, y_6	WRITEBUFFER ← F, DMAR ← DMAR + 1	
	$q_{18}t_6$	y_{12}	OUT ← WRITEBUFFER	
	$q_{18}t_7$	y_9	T ← 0	

Table 29. Six Data Word Load Instruction

Mnemonic	q_xt_x	Control Signal	Micro-Operation	Function
LOADALL addr	$q_{46}t_3$	y_{13}, y_6	A ← READBUFFER, DMAR ← DMAR + 1	load six data words to register A to F
	$q_{46}t_4$	y_{10}	READBUFFER ← INP	
	$q_{46}t_5$	y_{14}, y_6	B ← READBUFFER, DMAR ← DMAR + 1	
	$q_{46}t_6$	y_{10}	READBUFFER ← INP	
	$q_{46}t_7$	y_{15}, y_6	C ← READBUFFER, DMAR ← DMAR + 1	
	$q_{46}t_8$	y_{10}	READBUFFER ← INP	
	$Q_{46}t_9$	y_{16}, y_6	D ← READBUFFER, DMAR ← DMAR + 1	

Table 29. (continued)

	$q_{46}t_{10}$	y_{10}	READBUFFER ← INP	
	$q_{46}t_{11}$	y_{17}, y_6	E ← READBUFFER, DMAR ← DMAR + 1	
	$q_{46}t_{12}$	y_{10}	READBUFFER ← INP	
	$q_{46}t_{13}$	y_{18}	F ← READBUFFER	
	$q_{46}t_{14}$	y_9	T ← 0	

Table 30. Move Instruction

Mnemonic	q_xt_x	Control Signal	Micro-Operation	Function
MOV A,F	$q_{19}t_3$	y_{25}	A ← F	move A to F
	$q_{19}t_4$	y_9	T ← 0	
MOV B,F	$q_{20}t_3$	y_{26}	B ← F	move B to F
	$q_{20}t_4$	y_9	T ← 0	
MOV C,F	$q_{21}t_3$	y_{27}	C ← F	move C to F
	$q_{21}t_4$	y_9	T ← 0	
MOV D,F	$q_{22}t_3$	y_{28}	D ← F	move D to F
	$q_{22}t_4$	y_9	T ← 0	
MOV E,F	$q_{23}t_3$	y_{29}	E ← F	move E to F
	$q_{23}t_4$	y_9	T ← 0	
MOV F,B	$q_{24}t_3$	y_{30}	F ← B	move F to B
	$q_{24}t_4$	y_9	T ← 0	
MOV F,C	$q_{25}t_3$	y_{31}	F ← C	move F to C
	$q_{25}t_4$	y_9	T ← 0	
MOV F,D	$q_{26}t_3$	y_{32}	F ← D	move F to D
	$q_{26}t_4$	y_9	T ← 0	
MOV F,E	$q_{27}t_3$	y_{33}	F ← E	move F to E
	$q_{27}t_4$	y_9	T ← 0	

Table 31. Integer Instruction

Mnemonic	q_xt_x	Control signal	Micro-Operation	Function
INC A	$q_{28}t_3$	y_{34}	A ← A + 1	increment A
	$q_{28}t_4$	y_9	T ← 0	
INC B	$q_{29}t_3$	y_{35}	B ← B + 1	increment B
	$q_{29}t_4$	y_9	T ← 0	
DEC A	$q_{30}t_3$	y_{36}	A ← A – 1	decrement A
	$q_{30}t_4$	y_9	T ← 0	
DEC B	$q_{31}t_3$	y_{37}	B ← B – 1	decrement B
	$q_{31}t_4$	y_9	T ← 0	
CPL A	$q_{32}t_3$	y_{38}	A ← ¬A	complement A
	$q_{32}t_4$	y_9	T ← 0	
CPL B	$q_{33}t_3$	y_{39}	B ← ¬B	complement B
	$q_{33}t_4$	y_9	T ← 0	
ADD A,B	$q_{34}t_3$	y_{40}	A ← A + B	add A and B and transfer the sum to A
	$q_{34}t_4$	y_9	T ← 0	

Table 31. (continued)

ADDC A,B	$q_{35}t_3$ $q_{35}t_4$	Y_{41} Y_9	A ← A + B + 1 T ← 0	add A and B + Carry and transfer sum to A
SUB A,B	$q_{36}t_3$ $q_{36}t_4$	Y_{42} Y_9	A ← A – B T ← 0	subtract A - B and transfer difference to A
SUBB A,B	$q_{37}t_3$ $q_{37}t_4$	Y_{43} Y_9	A ← A – B – 1 T ← 0	subtract A - B – borrow, transfer difference to A
OR A,B	$q_{40}t_3$ $q_{40}t_4$	Y_{46} Y_9	A ← A ∨ B T ← 0	load A with A OR B
EXOR A,B	$q_{41}t_3$ $q_{41}t_4$	Y_{47} Y_9	A ← A ⊕ B T ← 0	load A with A XOR B
AND A,B	$q_{42}t_3$ $q_{42}t_4$	Y_{48} Y_9	A ← A ∧ B T ← 0	load A with A AND B
CMP A,B	$q_{43}t_3$ $q_{43}t_4$	Y_{49} Y_9	CMP A,B (A-B) T ← 0	compare A and B
ADDI A,addr	$q_{57}t_3$ $q_{57}t_4$	Y_{53} Y_9	A ← A + addr T ← 0	add memory content (address) to register A
SUBI A,addr	$q_{58}t_3$ $q_{58}t_4$	Y_{54} Y_9	A ← A – addr T ← 0	subtract memory content (address) to register A
ANDI A,addr	$q_{60}t_3$ $q_{60}t_4$	Y_{56} Y_9	A ← A ∧ addr T ← 0	register A AND memory content (address)
ORI A,addr	$q_{61}t_3$ $q_{61}t_4$	Y_{57} Y_9	A ← A ∨ addr T ← 0	register A OR memory content (address)
EXORI A,addr	$q_{62}t_3$ $q_{62}t_4$	Y_{58} Y_9	A ← A ⊕ addr T ← 0	register A EXOR memory content (address)
MUL A,B	$q_{38}t_3$ $q_{38}t_4$	Y_{44} Y_9	A ← B[31:16] * B[15:0] T ← 0	load A with product of low- and high-bytes of B
MUL C,B	$q_{39}t_3$ $q_{39}t_4$	Y_{45} Y_9	C ← B[31:16] * B[15:0] T ← 0	load C with product of low- and high-Bytes of B
MULTI A,addr	$q_{59}t_3$ $q_{59}t_4$	Y_{55} Y_9	A ←addr[31:16] * addr[15:0] T ← 0	load A with product of low- and high-Bytes of memory (address)
MUL64AB addr	$q_{47}t_3$ $q_{47}t_4$ $q_{47}t_5$ $q_{47}t_6$ $q_{47}t_7$ $q_{47}t_8$	y_{50} y_{51}, y_{11} y_{12} y_6, y_{52}, y_{11} y_{12} y_9	MREG64 ← A[31:0] * B[31:0] WRITEBUFFER ← REG[31 :0] OUT ← WRITEBUFFER DMAR ← DMAR + 1, WRITEBUFFER ← REG[63 :32] OUT ← WRITEBUFFER T ← 0	multiply register A and B, and write product to memory (address and address+1)

Table 32. Branch Instruction

Mnemonic	q_xt_x	Control signal	Micro-Operation	Function
JMP addr	$q_{44}t_3$ $q_{44}t_4$	y_5 y_9	PC ← DMAR T ← 0	jump unconditioned to address
RET addr	$q_{45}t_3$	y_5	PC ← DMAR	return unconditioned to address
JC addr	$q_{48}t_3$ $q_{48}t_4$	y_5 y_9	PC ← DMAR, if C T ← 0	jump if CARRY = 1
JNC addr	$q_{49}t_3$ $q_{49}t_4$	y_5 y_9	PC ← DMAR, if ¬C T ← 0	jump if CARRY = 0
JZ addr	$q_{50}t_3$ $q_{50}t_4$	y_5 y_9	PC ← DMAR, if Z T ← 0	jump if ZERO = 1
JNZ addr	$q_{51}t_3$ $q_{51}t_4$	y_5 y_9	PC ← DMAR, if ¬Z T ← 0	jump if ZERO = 0
JV addr	$q_{52}t_3$ $q_{52}t_4$	y_5 y_9	PC ← DMAR, if V T ← 0	jump if OVERFLOW = 1
JNV addr	$q_{53}t_3$ $q_{53}t_4$	y_5 y_9	PC ← DMAR, if ¬V T ← 0	jump if OVERFLOW = 0
JS addr	$q_{69}t_3$ $q_{69}t_4$	y_5 y_9	PC ← DMAR, if S T ← 0	jump if SIGN = 1
JNS addr	$q_{70}t_3$ $q_{70}t_4$	y_5 y_9	PC ← DMAR, if ¬S T ← 0	jump if SIGN = 0
CALL addr	$q_{54}t_3$ $q_{54}t_4$ $q_{54}t_5$ $q_{54}t_6$	y_{79} y_{79}, y_{80} y_5 y_9	STP.sel STP.sel, STP.inc PC ← DMAR T ← 0	jump and save processor state at the stack
RET	$q_{55}t_3$ $q_{55}t_4$ $q_{55}t_5$	y_{81}, y_{82} y_{83}, y_5 y_9	STR.sel, STP.dec PC.sel, PC ← DMAR T ← 0	return to address – loaded from stack

Table 33. NOP Instruction

Mnemonic	q_xt_x	Control Signal	Micro-operation	Function
NOP	$q_{56}t_3$	y_9	T ← 0	no operation

Table 34. Floating Point Instruction

Mnemonic	$q_x t_x$	Control Signal	Micro-Operation	Function
ADDF A,B	$q_{63}t_3$	y_{59}	A ← A + B (L1)	add float numbers in registers A and B (4 cycles), transfer result to A
	$q_{63}t_4$	y_{60}	A ← A + B (L2)	
	$q_{63}t_5$	y_{61}	A ← A + B (L3)	
	$q_{63}t_6$	y_{62}	A ← A + B (L4)	
	$q_{63}t_7$	y_{84}	A ← A + B (res)	
	$q_{63}t_8$	y_9	T ← 0	
SUBF A,B	$q_{64}t_3$	y_{63}	A ← A – B (L1)	subtract float numbers in registers A and B (4 cycles), transfer result to A
	$q_{64}t_4$	y_{64}	A ← A – B (L2)	
	$q_{65}t_5$	y_{65}	A ← A – B (L3)	
	$q_{64}t_6$	y_{66}	A ← A – B (L4)	
	$q_{64}t_7$	y_{85}	A ← A – B (res)	
	$q_{64}t_8$	y_9	T ← 0	
MULTF A, B	$q_{65}t_3$	y_{67}	A ← A * B (L1)	multiply float numbers in registers A and B (2 cycles), transfer result to A
	$q_{65}t_4$	y_{68}	A ← A * B (L2)	
	$q_{65}t_5$	y_{86}	A ← A * B (res)	
	$q_{65}t_6$	y_9	T ← 0	
ADDFI A,addr	$q_{66}t_3$	y_{69}	A ← A + addr (L1)	add float number in register A and memory (address) (4 cycles)
	$q_{66}t_4$	y_{70}	A ← A + addr (L2)	
	$q_{66}t_5$	y_{71}	A ← A + addr (L3)	
	$q_{66}t_6$	y_{72}	A ← A + addr (L4)	
	$q_{66}t_7$	y_{87}	A ← A + addr (res)	
	$q_{66}t_8$	y_9	T ← 0	
SUBFI A, addr	$q_{67}t_3$	y_{73}	A ← A – addr (L1)	subtract float number in register A and memory (address) (4 cycles)
	$q_{67}t_4$	y_{74}	A ← A – addr (L2)	
	$q_{67}t_5$	y_{75}	A ← A – addr (L3)	
	$q_{67}t_6$	y_{76}	A ← A – addr (L4)	
	$q_{67}t_7$	y_{88}	A ← A – addr (res)	
	$q_{67}t_8$	y_9	T ← 0	
MULTFI A, addr	$q_{68}t_3$	y_{77}	A ← A * addr (L1)	multiply float number in register A and memory (address) (2 cycles)
	$q_{68}t_4$	y_{78}	A ← A * addr (L2)	
	$q_{68}t_5$	y_{89}	A ← A * addr (res)	
	$q_{68}t_6$	y_9	T ← 0	

A.4 Pipeline Processors DLX32/64fpu_p

The architecture of these processors is based on a modified DLX-scheme by Hennessey and Patterson, which was described in [39]. The Harvard-architecture contains an instruction- and a data-RAM. The design is partitioned into control- (CP) and datapath (DP). The CP has a full accessibility to all control-signal. The DP contains a fast multiplier, a integer ALU and a floating point unit for single – respective double precision signed addition or subtraction (IEEE standard 754). Furthermore, the DP contains a register-file with 32 fix-point- or 16 floating-point registers (64-bit for DLX64).
The instruction-set is derived from [39]. An interrupt-handling functionality is not considered yet. The extension for this feature is easily implementable, because according lines exists (e.g. `MA_ERR` from FPU). Implemented instructions are listed in the following table:

Table 35. DLX32/64fpu_p Instruction Set

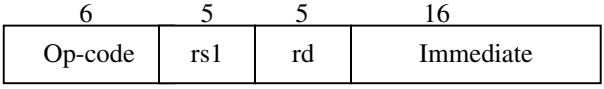
Instruction	Type	Meaning
ADD	Integer	Addition of two register-contents
ADDD	Floating point	Addition of two register-contents
ADDI	Integer	Addition of a register-content and an immediate
SUB	Integer	Subtraction of two register-contents
SUBD	Floating point	Subtraction of two register-contents
SUBI	Integer	Subtract an immediate from a register content
MUL	Integer	Multiplication of two register-contents
AND	Integer	AND-operation of two register-contents
ANDI	Integer	AND-operation of a register-content with an immediate
OR	Integer	OR-operation of two register-contents
ORI	Integer	OR-operation of a register-content and an immediate
XOR	Integer	Exclusive-OR-operation of two register-contents
XORI	Integer	Exclusive-OR-operation of a register-content and an immediate
SLL	Integer	One-bit logical left-shift of a register content
SRL	Integer	One-bit logical right-shift of a register content
SRA	Integer	One-bit arithmetical right-shift of a register content
J	Control	Jump PC-relative with 26-bit immediate
JR	Control	Jump register-direct
JAL	Control	Jump PC-relative with 26 bit immediate and write PC in register R31
JALR	Control	Jump register-direct and write PC in register R31
BEQZ	Control	Branch PC-relative with 16-bit immediate if register-content = zero
BNEZ	Control	Branch PC-relative with 16-bit immediate if register-content ≠ zero
LIW	Transport int.	Load least-significant 32-bit of a registers
LI	Transport int.	Load a 64-bit word of a registers
LD	Transport int.	Load a 64-bit word of a registers

Table 35. (continued)

SIW	Transport int.	Store least-significant 32-bit of a registers
SI	Transport int.	Store 64-bit of a registers
SD	Transport fp	Store 64-bit of a registers
MOVI	Transport int.	Move register content to another register
MOVD	Transport fp	Move register content to another register
NOP	Control	No operation
HALT	Control	Processor halt

Instructions of DLX32/64fpu_p processors have a static instruction length. They support register-direct, displacement and immediate address-modes. The instruction-formats are organized as follows:

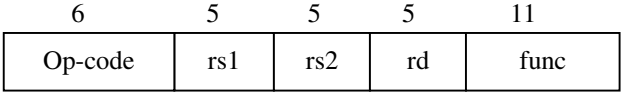
I – Type-instructions



Load and store of words and double-words
ALU-operations with Immediate (rd ← rs1 op Immediate)

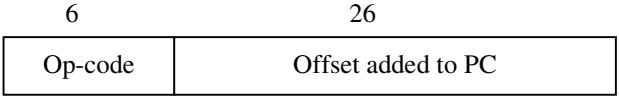
Conditional branch instructions
Jump register, Jump and Link Register
(rd=0, rs1= Destination, Immediate=0)

R – Type- instructions



Register-register-ALU-operation: rd ← rs1 func rs2
Data operations: Add, Sub, etc.

J – Type- instructions



Jump, jump and link

The pipeline processor has the following stages according Fig. 49:

1. Instruction fetch (IF)
2. Instruction decode and register fetch (ID)
3. Execution and address calculation (EXE)
4. Memory access (MEM)
5. Write back (WB)

The currently implemented duration for separate stages is organized as follows:

IF: 3 cycles
ID: 3 cycles
EXE: integer ALU, multiplier – 2cycles / FPU – 6 cycles
MEM: 6 cycles
WB: 2 cycles

Micro-instructions-sequences are outlined in the following table:

Table 36. DLX32/64fpu_p Micro-Instruction Sequences

Instr	IF	ID	EXE	MEM	WB
LIW	MAR←PC IR←M[MAR]	A←Src.1 B←Src.2	C←A+IMM	MAR2←C MD_LOW←M[MAR2]	Dest. ←MD_LOW
LI	MAR←PC IR←M[MAR]	A←Src.1 B←Src.2	C←A+IMM (integer)	MAR2←C MD_HIGH←M[MAR2] MAR←MAR+1 MD_LOW←M[MAR2]	Destination←MD
LD	MAR←PC IR←M[MAR]	A←Src.1 B←Src.2	C←A+IMM (integer)	MAR2←C MD_HIGH←M[MAR2] MAR←MAR+1 MD_LOW←M[MAR2]	Destination←MD
SIW	MAR←PC IR←M[MAR]	A←Src.1 B←Src.2	C←A+IMM (integer) B1←B	MAR2←C MD←B1 M[MAR2]←MD_LOW	
SI	MAR←PC IR←M[MAR]	A←Src.1 B←Src.2	C←A+IMM (Integer) B1←B	MAR2←C MD←B1 M[MAR2] ←MD_HIGH MAR←MAR+1 M[MAR2] ←MD_LOW	
SD	MAR←PC IR←M[MAR]	A←Src.1 B←Src.2	C←A+IMM (Integer) B1←B	MAR2←C MD←B1 M[MAR2] ←MD_HIGH MAR←MAR+1 M[MAR2] ←MD_LOW	
MOVI	MAR←PC IR←M[MAR]	A←Src.1 B←Src.2	C←A	C1←C	Destination←C1
MOVD	MAR←PC IR←M[MAR]	A←Src.1 B←Src.2	C←A	C1←C	Destination←C1
ADD	MAR←PC IR←M[MAR]	A←Src.1 B←Src.2	C←A+B (integer)	C1←C	Destination←C1
ADDD	MAR←PC IR←M[MAR]	A←Src.1 B←Src.2	C←A+B (fl. point)	C1←C	Destination←C1
ADDI	MAR←PC IR←M[MAR]	A←Src.1 B←Src.2	C←A+IMM (integer)	C1←C	Destination←C1
SUB	MAR←PC IR←M[MAR]	A←Src.1 B←Src.2	C←A-B (integer)	C1←C	Destination←C1

Table 36. (continued)

SUBD	MAR←PC IR←M[MAR]	A←Src.1 B←Src.2	C←A-B (fl. point)	C1←C	Destination←C1
SUBI	MAR←PC IR←M[MAR]	A←Src.1 B←Src.2	C←A-IMM (integer)	C1←C	Destination←C1
MUL	MAR←PC IR←M[MAR]	A←Src.1 B←Src.2	C←A*B (integer)	C1←C	Destination←C1
AND	MAR←PC IR←M[MAR]	A←Src.1 B←Src.2	C←A ∩ B (integer)	C1←C	Destination←C1
ANDI	MAR←PC IR←M[MAR]	A←Src.1 B←Src.2	C←A ∩ IMM (integer)	C1←C	Destination←C1
OR	MAR←PC IR←M[MAR]	A←Src.1 B←Src.2	C←A ∪ B (integer)	C1←C	Destination←C1
ORI	MAR←PC IR←M[MAR]	A←Src.1 B←Src.2	C←A ∪ B (integer)	C1←C	Destination←C1
XOR	MAR←PC IR←M[MAR]	A←Src.1 B←Src.2	C←A ⊕ B (integer)	C1←C	Destination←C1
XORI	MAR←PC IR←M[MAR]	A←Src.1 B←Src.2	C←A ⊕ B (integer)	C1←C	Destination←C1
SLL	MAR←PC IR←M[MAR]	A←src.1 B←Src.2	C←SLL A	C1←C	Destination←C1
SRL	MAR←PC IR←M[MAR]	A←Src.1 B←Src.2	C←SRL A	C1←C	Destination←C1
SRA	MAR←PC IR←M[MAR]	A←Src.1 B←Src.2	C←SRA A	C1←C	Destination←C1
BEQZ	MAR←PC IR←M[MAR]	A←Src.1 B←Src.2	C←PC + IMM (integer) Compare←B	PC←C if zero	
BNEZ	MAR←PC IR←M[MAR]	A←Src.1 B←Src.2	C←PC + IMM (integer) Compare←B	PC←C if not zero	
J	MAR←PC IR←M[MAR]	A←Src.1 B←Src.2	C←PC + IMM (integer)	PC←C	
JR	MAR←PC IR←M[MAR]	A←Src.1 B←Src.2	C←A (integer)	PC←C	
JAL	MAR←PC IR←M[MAR]	A←Src.1 B←Src.2	C←PC + IMM (integer) B1←PC	PC←C MD←B1	Destination←MD
JALR	MAR←PC IR←M[MAR]	A←Src.1 B←Src.2	C←A (integer) B1←PC	PC←C MD←B1	Destination←MD
NOP	MAR←PC IR←M[MAR]	A←Src.1 B←Src.2			
HALT	MAR←PC IR←M[MAR]				

Abbreviations, Symbols and Identifiers

ADR	Application driven reduction
ASIP	Application specific instruction set processor
ASP	Architecture specific partitioning
ALU	Arithmetic logic unit
BCP	Berger (code) check prediction
BIST	Built-in self-test
BISC	Built-in self-check
CISC	Complex instruction set computers
CL	Control logic
CLA	Carry-look-ahead (adder)
CMOS	Complementary MOS
COTS	Commercial of-the-shelf
CP	Control path
CPI	Clocks per instruction
CPLD	Complex programmable logic device
CPP	Cross-parity prediction
CPU	Central processing unit
CSP	Control signal prediction
CW	Control word
DP	Data path
DSP	Digital signal processor
EEPROM	Electrical erasable PROM
ENIAC	Electronic numerical integrator and calculator
FIFO	First-in first out
FT	Fault-tolerant
FS	Fault secure
FSM	Finite state machine
HDL	Hardware description language
IC	Integrated circuit
ID	Instruction decoder
IP	Intellectual property
IR	Instruction register
JTAG	Joint Test Action Group
LFSR	Left feedback shift register
MAR	Memory address register
MBR	Memory buffer register
MIMD	Multiple instruction Stream, Multiple Data Stream
MISD	Multiple instruction Stream, Single Data Stream
MISR	Multiple input shift register
µOp	Micro-operation
MOS	Metal-oxide-semiconductor (transistor)

PC	Program counter
PROM	Programmable read-only memory
RAM	Read access memory
RAPID	Reusable Application Specific Intellectual Property Developers
RISC	Reduced instruction set computers
RB	Rollback
ROM	Read-only memory
RTL	Register transfer level
Op-code	Operation code respective binary coded instruction
sa0	Stuck-at-zero (fault)
sa1	Stuck-at-one (fault)
SISD	Single instruction stream, single data stream
SIMD	Single instruction stream, multiple data stream
SOB	System-on-board
SOC	System-on-chip
SP	Stack pointer
TMR	Triple-modular redundancy
Vdd	Power supplier or logic ′1′
UDL	User defined logic
VHDL	VHSIC hardware description language
VHSIC	Very high speed integrated circuit
VLIW	Very long instruction word (processor)
VSIA	Virtual Socket Interface Alliance
Vss	Ground or logic ′0′

List of Figures

List of Tables

References

1. M. Abramovici, M.A. Breuer, A.D. Friedman: Digital System Testing and Testable Design, IEEE Press, New York, ISBN 0-7803-1062-4, 1990
2. K.-T. Cheng, S. Dey, M. Rodgers, K. Roy: Test Challenges for Deep Sub-Micron Technologies, DAC2000, Los Angeles (CA), pp. 142-149
3. R.K. Gupta, Y. Zorian: Introducing Core-Based System Design, IEEE Design & Test Of Computers Oct.-Dec. 1997, pp. 15-25
4. Y. Zorian, E.J. Marinissen, S. Dey: Testing Embedded-Core Based System Chips, ITC 1998, pp. 130-143
5. Y. Zorian, E.J. Marinissen: System Chip Test: How Will It Impact Your Design?, DAC2000, Los Angeles, CA
6. IEEE Computer Society: IEEE Standard Test Access Port and Boundary-Scan Architecture – IEEE Std. 1149.1-1990, IEEE, New York, 1990
7. IEEE P1500 Web Site: http://grouper.ieee.org/groups/1500
8. A.P. Ströle, H.J. Wunderlich: TESTCHIP: A Chip for Weighted Random Pattern Generation, Evaluation and Test Control, IEEE Journal of Solid State Circuits, Vol. 26, No. 7, July 1991, pp. 1056-1063
9. A. Auer, R. Kimmelmann: Schaltungstest mit Boundary Scan, Hüthig Verlag, ISBN 3-7785-2519
10. D.K. Pradhan: Fault-Tolerant Computer System Design, Prentice Hall, NJ, 1996, ISBN: 0-13-057887-8
11. U. Gläser: Mehrebenen-Testgenerierung für synchrone Schaltwerke, GMD-Bericht Nr. 235, 1994, Oldenburg-Verlag, ISBN 3-486-23193-6
12. U. Gläser, U. Hübner, H.T. Vierhaus: Mixed Level Hierarchical Test Generation for Transition Faults and Overcurrent Related Defects, Proc. International Test Conference, Baltimore, MD, USA, Sept. 1992, pp. 21-29
13. U. Gläser, H. T. Vierhaus: Mixed-Level Test Generation for Synchronous Sequential Circuits using the FOGBUSTER Algorithm, IEEE Transactions of Computer-Aided Design of ICs, Vol. 15, No, 4, April 1996, pp. 410-423
14. W. Meyer, R. Camposano: Fast Hierarchical Multi-Level Fault Simulation of Sequential Circuits with Switch-Level Accuracy, 30th ACM/IEEE DAC, 1993, p. 515 – 519
15. M. Pflanz, H.T. Vierhaus: Generating Reliable Embedded Processors, IEEE MICRO Sept./Oct. 1998, pp. 33-41
16. M. Pflanz, C. Rousselle, H.T. Vierhaus: Possibilities and Limitations of Self-Test and Functional Backup for Standard Processor Cores in Embedded Applications, European Test Workshop, May 1999, Constance, Germany
17. M. Pflanz, H.T. Vierhaus: An Efficient On-Line-Test and Recover Scheme for Embedded Processors, 10th European Workshop on Dependable Computing, May 1999, Vienna, pp. 63-69
18. M. Pflanz, F. Pompsch, H. Hennig, H.T. Vierhaus: Efficient Backup Schemes for Processors in Embedded Systems, European Material Conference (E-MRS), June 1999, Symposium M, paper M-II.3 and in Journal Solid-State Electronics, No. 44, 2000, pp. 791-796
19. M. Pflanz, F. Pompsch, H.T. Vierhaus: An Efficient On-Line-Test and Back-up Scheme for Embedded Processors, Proceedings Int. Test Conference, ITC'99, Atlantic City, MA, pp. 964-972

20. M. Pflanz, C. Galke, H.T. Vierhaus: A New Method for On-Line State Machine Observation for Embedded Microprocessors, Proc. IEEE Int. High Level Design Validation and Test Workshop, Nov. 2000, Berkeley, CA, pp. 34-39
21. M. Pflanz, K. Walther, H.T. Vierhaus: On-line Error Detection Techniques for Dependable Embedded Processors with High Complexity, Int. On-line Test Workshop (IOLTW'01), July, 2001, Taormina, Italy
22. B.W. Johnson: Design and Analysis of Fault-Tolerant Digital Systems, Addison-Wesley Publishing Company, Reading, Mass., 1989, p. 173
23. P.C.Li, T.K.Young: Electromigration: The time bomb in deep-submicron Ics, IEEE Spectrum, Sept. 1996, pp.75-78
24. K.W. Li, J.R. Armstrong, J.G. Tront: An HDL Simulation of the Effects of Single Event Upsets On microprocessor Program Flow, IEEE Trans. on Nuclear Science, Vol. NS-31, No. 6, Dec. 1984
25. M. Nicolaidis: Online Testing for VLSI: State of the Art and Trends; Integration the VLSI Journal, No. 28, 1998, pp. 197-209
26. A.Benson, M.Rebaudengo, L.Impagliazzo, P.Marmo: Fault List Collapsing for Fault Injection Experiments, Annual Reliability and Maintainability Symposium, 1998, pp.383-388
27. R.D. Eldred: Test Routines Based on Symbolic Logic Statements, Journal of the ACM, January 1959, pp. 33-36
28. R.L. Wadsack: Fault Modelling and Logic Simulation in CMOS and MOS Integrated Circuits, Bell Systems Technical Journal, May-June 1978, pp. 1449-1474
29. R.K. Gulati, G.F. Hawkins: Iddq Testing of VLSI Circuits, Kluwer Academic Publishers, Boston, 1993
30. J.M. Acken: Testing for Bridging Faults (Shorts) in CMOS Circuits, 1983 Design Automation Conference, pp. 717-718, 1983
31. K.J. Lee, M.A. Breuer: On detecting Single and Multiple Bridging Faults in CMOS Circuits Using Current Supply Monitoring Method, 1990 International Conference on Circuits and Systems, May 1990
32. T.M Storey, W. Maly: CMOS Bridging Fault Detection, 1990 International Test Conference, Sept. 1990, pp. 842-851
33. H.T. Vierhaus, W. Meyer, U. Gläser: CMOS Bridges and Resistive Transistor Faults: IDDQ versus Delay Effects, Int. Test Conference 1993, Baltimore, USA, pp. 83-91
34. P.K. Lala: Digital circuit Testing and Testability, Academic Press, ISBN 0-12-434330-9
35. M. Rimén, J. Ohlson: A Study of the Error Behavior of a 32-bit RISC Subjected to Simulated Transient Fault Injection, Proc. Of the Int. Test Conference 1992, Baltimore, USA, pp. 696-704
36. C. Rousselle: Entwicklung eines RT-/Logik-Simulators auf der Basis von hierarchischen Schaltungsbeschreibungen in VHDL, Master thesis at the CE Dept., BTU Cottbus, Germany, October 2000
37. A. Behling: Entwicklung eines minimierenden Fehlerlistengenerators, Master thesis at the CE Dept., BTU Cottbus, Germany, October 2000
38. C. Rousselle, M. Pflanz, A. Behling, T. Mohaupt, H.T. Vierhaus: A Register-Transfer-Level Fault Simulator for Permanent and Transient Faults in Embedded Processors, DATE'01, Munich, March 2001, Germany
39. D.A. Patterson, J.L. Hennessy: Computer Organization and Design – The Hardware/Software Interface; Morgan Kaufmann Publishers Inc., San Francisco, CA, 1998
40. http://ei.cs.vt.edu/~history/Zuse.html
41. M.M. Mano: Digital Logic and Computer Design, Prentice-Hall Inc., New Jersey, 1979, ISBN 0-13-214510-3
42. M.J. Flynn: Very high-speed computing systems, Proc. IEEE 54:12, Dec. 1966, pp. 396, 591
43. A. Bode: RISC-Architecturen, Reihe Informatik, Band 60, BI Wissenschaftsverlag, Mannheim, 1990, ISBN 3-411-14752-0

44. E.J. Marinissen, Y. Zorian, R. Kapur, T. Taylor, L. Whetsel: Towards a Standard for Embedded Core Test: An Example, Proc. ITC'99, Atlantic City, NJ, Oct. 1999, pp. 616-627
45. A.R. Weiss: The Standardization of Embedded Benchmarking: Pitfalls and Opportunities, Proc. IEEE Int. Conf. On Computer Design, ICCD'99, Austin, TX, Oct. 1999, pp. 492-498
46. M.F. Jacome, G. deVeciana: Design Challanges for New Application-Specific Processors, IEEE Design & Test of Computers, April-June 2000, pp. 40 - 50
47. A. Amendola, A. Benso, F. Corno, L. Impagliazzo, P. Prinetto, M. Rebaudengo, M. Sonza Reorda: Faulty Behavior Observation on a Microprocessor System through a VHDL Simulation-Based Fault Injection Experiment, IEEE EURO-VHDL'96, Geneva (Switzerland), September 1996
48. A. Benso, M. Rebaudengo, M. Sonza Reorda: Fault Injection for Embedded Microprocessor-based Systems; Journal of Universal Computer Science (Special Issue on Dependability Evaluation and Validation), 1998, Vol. 5, No. 5, pp. 693-711
49. J.A. Clark, D.K. Pradhan: Fault Injection – A Method for Validating Computer-System Dependability, IEEE Computer, June 1995, pp. 47-56
50. M. Gössel, S. Graf: Error Detection Circuits, McGraw-Hill Book Company, London, ISBN 0-07-707438-6, 1993
51. F.F. Sellers, M.J. Hsiao, L.W. Bernson: Error Detecting Logic for Digital Computers, McGraw-Hill Book Company, New York, 1968
52. R.H. Minero, A.J. Anello, R.G. Furey, L.R. Palounek: Checking by Pseudoduplication, US Patent 3660646,GO6F 11/00, 1972
53. N.T. Wing, E. Glen: Self Checking Arithmetic Unit, US Patent 4314350, GO6F 11/14
54. J.M. Berger: A Note on Error Detecting Codes for Asymmetric channels, Information and Control, vol. 4, pp 68-73, 1961
55. A. Morosow, V.V. Saposhnikov, Vl.V. Saposhnikov, M. Goessel: Self-Checking Circuits with Unidirectionally Independent Outputs Journal VLSI Design, Vol 5 No 4. pp. 333-345, 1998
56. S. Mitra, E. McCluskey: Which Concurrent Error Detection scheme to Choose?, International Test Conference, Atlantic City, NJ, USA, 2000, pp. 985-994
57. V. Otscheretnij, M. Goessel, Vl.V. Saposhnikov, V.V. Saposhnikov: Fault-Tolerant Self-dual Circuits with Error Detection by Parity- and Group Parity Prediction, Proc. 4th IEEE International On-Line Testing Workshop, pp.124-130, Capri, Italy, 1998
58. C. Zeng, N. Saxena, E.J. McCluskey: Finite State Machine Synthesis with concurrent Error Detection, Proc. ITC'99, Atlantic City, NJ, 1999, pp. 672-679
59. J-C. Lo, S. Thanawastien, T.R.N. Rao, M. Nicolaidis: An SFS Berger Check Prediction ALU and Its Application to Self-Checking Processor Designs, IEEE Trans. On CAD, Vol. 11, No. 4, April 1992, pp.525-540
60. Lattice™ Semiconductor Corporation: Macro Library Reference Manual, Version 4.00, 1996
61. R. Leveugle, T. Michel, G. Saucier: Design of Microprocessors with Built-In On-Line Test, Proc. IEEE 20th Int. Symp. On Fault-Tolerant Computing, Los Alamitos, CA, June 1990, pp. 450-456
62. M.A. Schuette, J.P. Shen: Processor Control Flow Monitoring Using Signatured Instruction Streams, IEEE Transactions on Computers, vol. C-36, No. 3, March 1987, pp. 264-275
63. G. Miremadi, J. Ohlson, M. Riemén, J. Karlsson: Use of Time and Address Signatures for Control Flow Checking, Dependable Computing for Critical Applications, No. 5, IEEE Computer Society, 1998, pp. 201-221
64. S. Hellebrand and H. Wunderlich, "An Efficient Procedure for the Synthesis of Fast Self-Testable Controller Structures," *Proc. Int'l Conf. Computer-Aided Design,* IEEE Computer Society Press, Los Alamitos, Calif., 1994, pp. 110-116
65. S.N. Hamilton, A. Hertwig, A. Orailoglu: Self Recovering Controller and Datapath Codesign, DATE 1999

66. E. Voskamp, W. Rosenstiel: Error Detection in Fault Secure Controllers using State Encoding, Proc. European Design & Test Conference, March 1996, Paris, pp. 200-204
67. P.K. Lala: Self-Checking and Fault-Tolerant Digital Design; Morgan Kaufmann Publishers, San Francisco, CA, 2000
68. URL = http://www.stratus.de/informationen/systemdesign.htm
69. URL = http://www.sun.com/smi/Press/sunflash/9606/sunflash.960624.4689.html
70. D.Powell, et al.Guards: a generic upgradable architecture for real-time dependable systems, LAAS-CNRS Rapport No. 98259, June 1998, Toulouse, France, 32p
71. C.F. Webb: S/390 microprocessor design, IBM Journal Res. Develop., Vol. 44, No. 6, November 2000, pp. 899-907
72. M. Rebaudengo, M. Sonza Reorda, M. Torchiano, M. Violante: Soft-error Detection through Software Fault-Tolerance techniques, IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems, Nov. 1999, Albuquerque, New Mexico, USA
73. R. Koo, S. Toueg: Checkpointing and Rollback-Recovery for Distributed Systems, IEEE Trans. on Software Engineering, Vol. SE-13, January 1987, pp. 23-31
74. Y. Tamir, M. Tremblay: High-Performance Fault-Tolerant VLSI Systems Using Micro Rollback, IEEE Trans. on Computers, Vol. 39, No. 4, April 1990, pp. 548-554
75. Y.Tamir, M. Liang, T. Lai, M. Tremblay: The UCLA Mirror Processor: A Builing Block for Self-Checking Self-Repairing Computing Nodes, Proc. 21th Intl. Symposium on Fault-Tolerant Computing (FTCS), 1991, Montreal, Canada
76. M. Nicolaidis, R.O. Duarte, S. Manich, J. Figueras: Fault-Secure Parity Prediction Arithmetic Operators, IEEE Design & Test of Computers, April-June, 1997
77. C. Galke: Rollback-Strategies for Processors with a Pipeline Structure, Master-Thesis (in German) at the CE Dept., BTU Cottbus, Germany, October 2000
78. M. Pflanz: Cross-Parity-Prediction, applied for a patent at the German Patent and Trademark Office (Deutsches Patent- und Markenamt), Febr., 2001, No.: 12070181