



NMRC RISC Core

Architectural Brief

Revision 1.1

431283.001D7

Module® and **NeuroMatrix®** are registered trademarks of JSC Research Center "Module". All other trademarks are the exclusive property of their respective owners.

Contents

1 General Description.....	4
2 RISC CORE	6
3 Memory Map	12
4 Interrupts.....	13
5 Brief Instruction Set Description.....	14
5.1 Instruction Formats.....	14
5.2 Addressing Modes.....	17
5.3 Address Register Modification Modes.....	18
5.4 PC Modification Modes.....	19
5.5 Condition Codes and Flags.....	20
5.6 Register Set.....	21
5.7 Scalar Arithmetic and Logic Operations	23

The 32/64-bit NeuroMatrix RISC Core (NMRC) has been designed as a powerful engine, that works as standalone as in cooperation with 32- or 64-bit co-processors (DSP co-processor for example). It allows easy way to build variety of high-performance application specific processors.

NeuroMatrix® RISC Core Key Features

Architecture

- original RC Module RISC architecture;
- 32- and (or) 64-bit data paths;
- advanced Harvard architecture;
- extended architecture for 32- or 64-bit co-processors.

Registers

- eight 32-bit general purpose registers;
- eight 32-bit address registers;
- two 32-bit timers;
- four 32-bit DMA channel counters;
- four 32-bit DMA channel address registers;
- four 64-bit DMA channel data registers;
- two 32-bit status registers;
- 32-bit program counter;
- possibility to use up to 16 user defined registers.

Address space

- 16-Gbyte address space;
- two address generators support up to two memory accesses per cycle.

Pipeline

- five-stage pipeline;
- 10 bypass paths to minimize the effect of pipeline latency on dependent operations

Instruction Set

- VLIW 32-bit instructions;
- 32-bit length allows up to four operations per instruction;
- load-store architecture;
- delayed branches to reduce pipeline disruption;
- possibility to add new instructions.

Execution time

- one instruction per clock.

Arithmetic operations

- two address adders;
- one ALU;
- one barrel shifter.

Multiply operations

- multiplications executed in 16 clocks (32 bits \times 32 bits \rightarrow 64 bits) on ALU;
- possibility to add 32 \times 32 array multiplier.

On-Core peripheral modules

- interrupt controller;
- two channel DMA controller;
- two free-running 32-bit timers.

Performance

- 50 MHz, 50 MIPS, 150 MOPS on 0.5 μ m @ 3.3V (silicon verified);
- 2-4 times faster than ARM7 at the same clock rate;
- approx. 35 Kgates;

Software development tools

- C compiler;
- assembler;
- linker;
- debugger (code simulator);
- Instruction Level Simulator;
- Source Code Debugger (command line and GUI for Windows95, NT);
- C Run Time Library.

The NMRC is intended for processing of 32-bit scalar data and of supporting 32- or 64-bit co-processors including DSP types. RISC Core block diagram is shown in Fig. 1-1. It comprises the following functional units:

- **RALU** - register and arithmetic/logic unit;
- **DAG1** and **DAG2** - primary and secondary data address generators;
- **PROGRAM SEQUENCER** - program sequencer;
- **CONTROL UNIT** - control unit;
- **TIMERS** - block of timers;
- **DMA CO-PROCESSOR** - DMA co-processor with two channels.

RISC Core can work with up to five external buses for rapid data exchange between the functional units:

- **LOCAL ADDRESS BUS** and **GLOBAL ADDRESS BUS** are used to transfer instruction addresses generated by RISC Core and data addresses generated by RISC Core in program mode and by DMA co-processor for memory access.
- **OUTPUT DATA BUS** is used to transfer data that have to be written to internal or external memory from RISC Core, DMA co-processor or application specific co-processor.
- **INPUT BUS #1** and **INPUT BUS #2** are used to transfer instructions and data fetched from memory to any of the main functional units. The bus **INPUT BUS #2** is exclusively occupied by scalar data and **INPUT BUS #1** is exclusively occupied by data for application specific co-processor in the normal mode. Data transfers via DMA mode and instruction transfers can use any available input bus.

Data move from one scalar register to another and immediate constant load from instruction buffer to scalar register are provided by means of programmable memory interface units using internal buses **OUTPUT DATA BUS** and **INPUT BUS #2**.

Internal buses **INPUT BUS #1**, **INPUT BUS #2** and **OUTPUT DATA BUS** are 64-bit wide. 32-bit or 64-bit transfer operation/cycle via these buses is possible. To increase efficiency of data moving between 32-bit scalar registers and between 32-bit registers and memory these registers are grouped into 64-bit register pairs. Also NMRC contains some 64-bit control registers. Thus it is possible to say that NMRC supports 64-bit transfer operations of scalar data.

Instruction fetch is performed by 64-bit words. Each word is a one 32-bit instruction with 32-bit immediate constant or two 32-bit instructions.

RISC Core comprises the following internal buses that intended for data exchange between RISC Core units:

- buses for the first and the second ALU operands;

- bus for the result of arithmetical logical operation or shift operation;
- buses for the first and the second AU1 operands;
- buses for the first and the second AU2 operands;
- program counter PC input bus;
- DMA co-processor input bus;
- DMA co-processor output bus.

All internal units and buses of RISC Core are 32-bit wide.

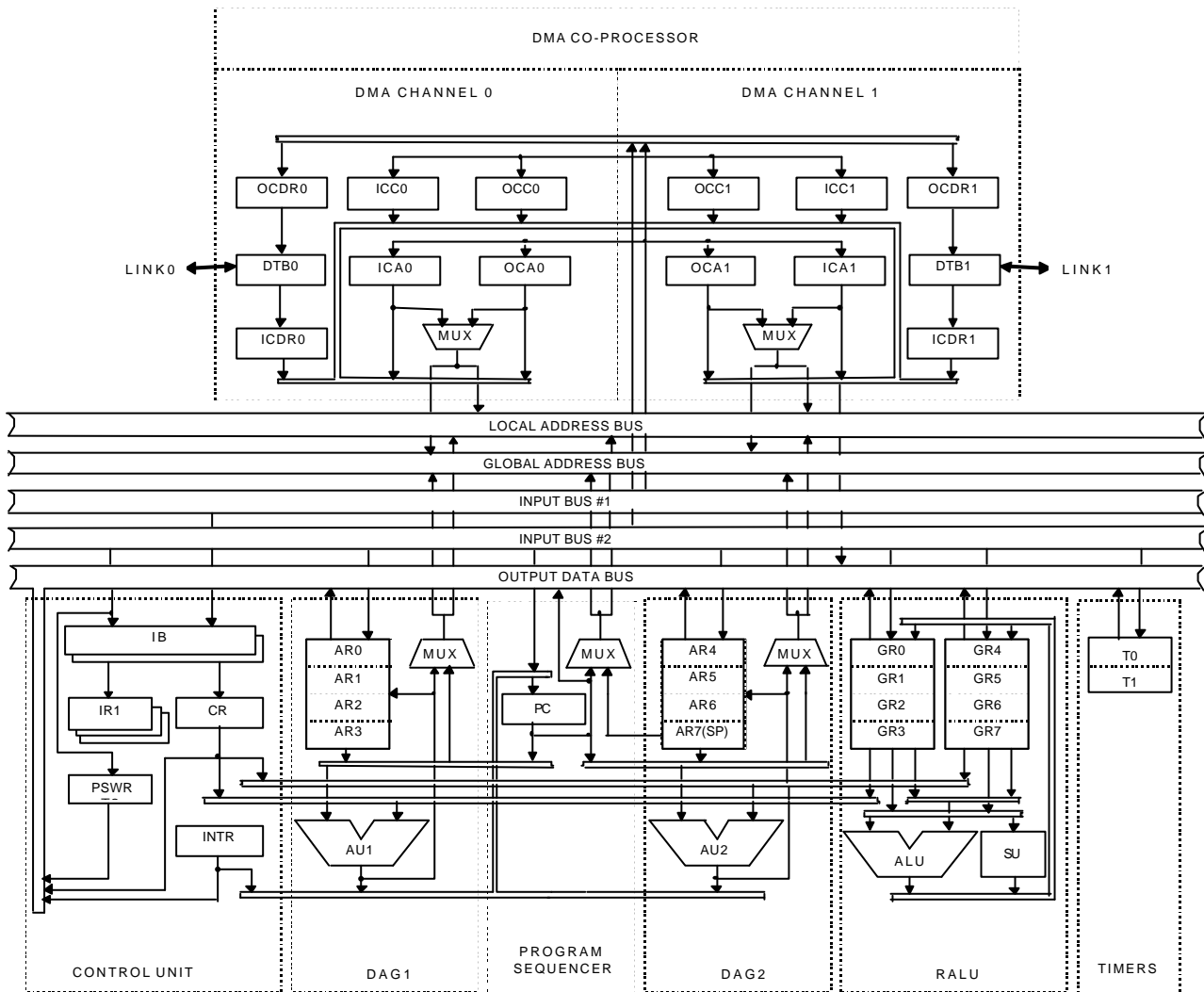


Fig. 1-1. NMRC Block Diagram

2.1 Register and Arithmetic/Logic Unit

Register and arithmetic/logic unit (RALU) is intended for storage of up to eight 32-bit scalar data and for execution of shift operations and single-operands and

double-operands arithmetical and logical operations over them. By execution of operations RALU forms some flags: zero result, negative result, carry and arithmetic overflow that can be fixed into program state word register for following reference by conditional branch instructions. The data stored in RALU also can be used as address or address displacement for some memory access or control instructions.

RALU comprises the following subblocks:

GR0, ... , GR7 - general purpose registers that are united into a register file. This register file has two input ports respectively connected to the *INPUT BUS #2* and to the *ALU* or shifter result bus and three output ports respectively connected to *OUTPUT DATA BUS* and to the first and the second *ALU* operand buses. Also the registers *GR0, ... , GR3* have an output connected to the second operand bus of *AU1*. That allows to use data stored in that registers when instruction or data address are formed at *DAG1* or when modification of address registers occurs at *DAG1*. Registers *GR4, ... , GR7* have an output connected to the second operand bus of *AU2*. That allows to use data stored in that registers when instruction or data addresses are formed at *DAG2* or when modification of address registers occurs at *DAG2*.

ALU - arithmetic/logic unit that is able to provide one of sixteen logical or up to sixteen arithmetical operations using data from one or two general purpose registers per one clock cycle. Arithmetical operations are conducted over two's complement 32-bit data.

SHIFTER - barrel shifter unit, that is able to provide left or right arithmetical or logical shift or rotate at any bit count for data placed from any general purpose register to the first *ALU* operand bus per one clock cycle.

2.2 Primary data address generator

Primary data address generator (*DAG1*) is intended for data address generation and instruction address generation for branch instructions. Also *DAG1* provides storage and modification of up to four 32-bit data addresses, branch addresses or address displacements.

DAG1 comprises the following subblocks:

AR0, ... , AR3 - address registers that are united into a register file that has two input ports respectively connected to *NMRC INPUT BUS #2* and to the *AU1* outputs and two output ports respectively connected to *OUTPUT DATA BUS* and to the first input operand bus of *AU1*.

AU1 - the first arithmetical unit to perform single operand or double operand arithmetical address calculations or modification of one of *DAG1* address registers. Data from *AR0, ... , AR3* or from *PC* can be used as the first operand and data from *GR0 - GR3* or 32-bit immediate constant can be used as the second operand.

MUX - address multiplexor to control address outputting from *AU1* output or from the first operand bus of *AU1* to address bus depending on the addressing mode. If *MSB* address value at multiplexor output is equal to 0, the address

from DAG1 is placed to *LOCAL ADDRESS BUS*, otherwise the address from DAG1 is placed to *GLOBAL ADDRESS BUS*.

2.3 Secondary data address generator

The secondary data address generator DAG2 is structurally and functionally very similar to DAG1. The main specific feature of DAG2 is that one of its address registers AR7 also can be used as a stack pointer SP for interrupt processing and subroutine handling instructions.

DAG2 comprises the following subblocks:

AR4, ... , AR7(SP) - address registers that are united into a register file that has two input ports respectively connected to *INPUT BUS #2* and to the AU2 output and two output ports respectively connected to *OUTPUT DATA BUS* and to the first input operand bus of AU2.

AU2 - the first arithmetical unit to perform single operand or double operand arithmetical address calculations or modification of one of DAG2 address registers. Data from AR4, ..., AR7 or from PC can be used as the first operand and data from GR4 - GR7 or 32-bit immediate constant can be used as the second operand.

MUX - address multiplexor that controls address outputting from AU2 output or from the first operand bus of AU2 address bus depending on addressing mode. If MSB address value at multiplexor output is equal to 0, address from DAG2 is placed to *LOCAL ADDRESS BUS*, otherwise the address from DAG2 is placed to *GLOBAL ADDRESS BUS*.

2.4 Program Sequencer

PROGRAM SEQUENCER is intended for forming the address of the next 32-bit instruction with 32-bit immediate constant or the address of the next pair of 32-bit instructions at linear program blocks when the next instruction address is defined from the current instruction address by an increment operation. Also PROGRAM SEQUENCER is used to form interrupt vector address and to modify stack pointer during call instructions.

PROGRAM SEQUENCER comprises the following subblocks:

PC - program counter that intended for storage of the address of a currently fetched 32-bit instruction with 32-bit immediate constant or a pair of 32-bit instructions and calculating of the next instruction or instruction pair address by post-incrementing by 2. By branch instructions the new address data feed PC from AU1 or AU2 output. When interrupt is processed, address data is taken from register INTR and when return from subroutine or return from interrupt operation occurs, address data is taken from *INPUT BUS #2*. PC output is connected to NMRC *OUTPUT DATA BUS* that makes it possible to read the value from PC to user program.

MUX - address multiplexor that controls address outputting from PC incremented by 2 or from stack pointer AR7(SP) to address bus. If MSB address value at multiplexor output is equal to 0 the address from PROGRAMM SEQUENCER is placed to *LOCAL ADDRESS BUS*, otherwise the address from DAG2 is placed to *GLOBAL ADDRESS BUS*.

2.5 Control Unit

CONTROL UNIT - performs the initial analysis and decoding of instructions that are fetched from external memory. This unit forms control signals to all functional units during pipeline instruction executing. It processes interrupt requests and performs request arbitration of the resources such as external and internal buses for other functional units.

CONTROL UNIT comprises the following subblocks:

IB - instruction buffer that provides store and preliminary analysis up to four 32-bit or two 64-bit instructions, fetched from external memory.

IR1, ... , IR3 - pipeline stages instruction registers.

CR – register for storage of the immediate constant of a current instruction.

PSWR - program state word register. This register is used for storage of the last RALU operation flags, interrupt mask and information of timers T0 and T1 control. This register is combined with the program counter PC and this register pair is stored at system stack when instruction call or interrupt occurs. This register are available for user program for reading and writing.

INTR - interrupt and DMA requests register. This register is used for storage of all requests for DMA, and all interrupt requests before they are processed. Register INTR are connected to *OUTPUT DATA BUS* and available for read only. Their input are connected to *INPUT BUS #2* and output are connected to *OUTPUT DATA BUS*. Also there is hardware support for mask based setting or resetting of any bit or bits of PSWR by single instruction.

2.6 Block of timers

Block of timers comprises the following subblocks:

T0, T1 –programmable timers. They are used to form interrupt signals in a programmable time range. Operation mode for each timer (single or periodical) is specified by a corresponding bits of PSWR register. T0 and T1 are available for user program for reading and writing. Their inputs are connected to *INPUT BUS #2* and outputs are connected to *OUTPUT DATA BUS*.

2.7 DMA CO-PROCESSOR

DMA CO-PROCESSOR has two identical DMA channels 0 and 1. It supports memory-to-memory data blocks transfers or transfers between NMRC and external device through links.

DMA CO-PROCESSOR comprises the following subblocks:

ICC0, ICC1 - counters for input channel 0 and 1 respectively.

ICA0, ICA1 - address registers for input channel 0 and 1.

OCC0, OCC1 - counters for output channel 0 and 1.

OCA0, OCA1 - address registers for output channel 0 and 1.

All registers ICC0, ICC1, ICA0, ICA1, OCC0, OCC1, OCA0, OCA1 are available for user program for reading and writing. Their inputs are connected to DMA co-processor input bus and outputs are connected to DMA co-processor output bus.

DIR0, DIR1 - data input registers for input channel 0 and 1 respectively. These registers are connected to DMA co-processor output bus and available for read only.

DOR0, DOR1 - data output registers for output channel 0 and 1 respectively. These registers are connected to DMA co-processor output bus and available for write only.

DTB0, DTB1 - data transmit buffers of DMA channels 0 and 1. These buffers are not program available.

MUX - multiplexors for selection of the address source for the DMA request.

The NMRC supports 32-bit internal address with 32-bit and 64-bit memory access. The total memory range of the RISC Core is 4Giga 32-bit words or 16Gbyte. This address space is divided into two equal parts: local and global (see Fig. 2-1). If address MSB is equal to zero then there is an access to local memory, if address MSB is equal to one then there is an access to global memory. The address LSB is used for access to 32-bit data. If address LSB is equal to zero the data from low memory word (bits 31-0) are used, if address LSB is equal to one the data from high memory word (bits 63-32) are used. While accessing 64-bit data or fetching an instruction the internal address LSB is ignored.

32-bit external memory access is available only for scalar instructions if specified source/destination register is 32-bit wide. If 64-bit register is specified the 64-bit data access is used.

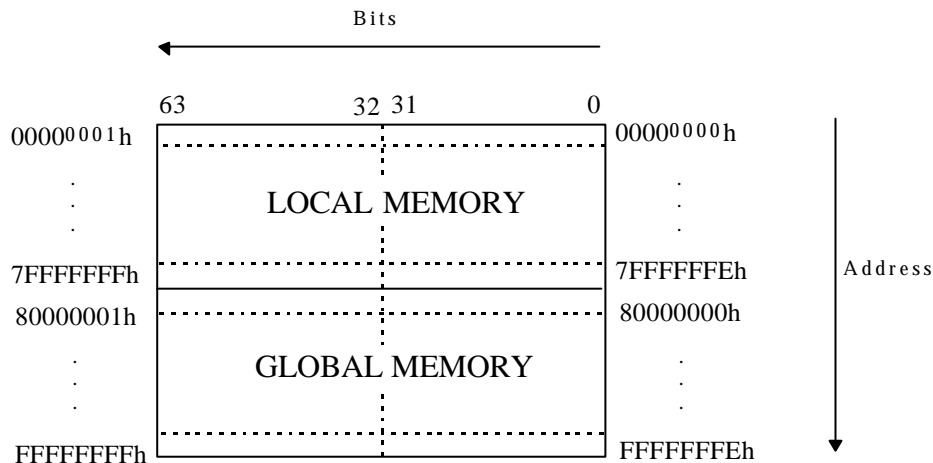


Fig. 2-1. Memory Map

The NMRC supports one external interrupt and 9 internal interrupts:

- two timer interrupts;
- arithmetic overflow during the operation at RISC Core interrupt;
- illegal or reserved instruction interrupt;
- four interrupts concerned with DMA input or output channel operation finished;
- trace interrupt.

These interrupts are represented in Table 4-1 with higher priority interrupt in the table top (in the case of multiple interrupt request at the same time the interrupt with higher priority will proceed first).

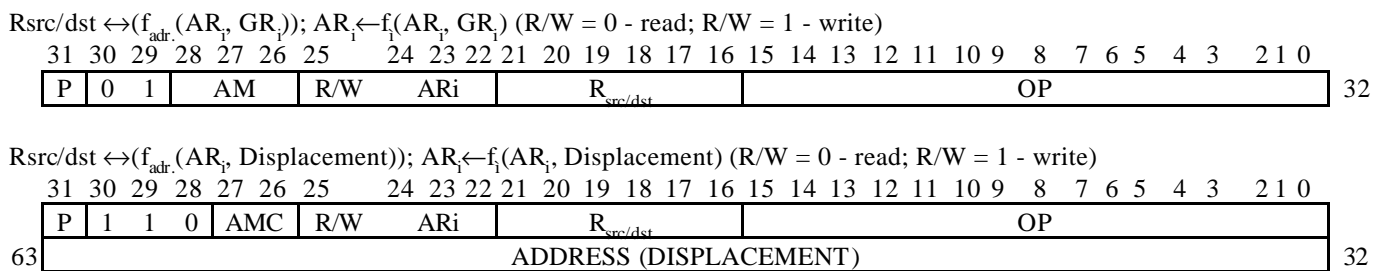
Table 4-1 NMRC Interrupts

1	Interrupt Source	Interrupt Starting Address
1.	System timer $\bar{O}0$	00000000h
2.	Arithmetic overflow during operation at RISC-Core	00000008h
3.	Illegal or reserved instruction	00000010h
4.	External interrupt	00000018h
5.	Input via DMA channel 1 is finished	00000020h
6.	Input via DMA channel 0 is finished	00000028h
7.	Output via DMA channel 1 is finished	00000030h
8.	Output via DMA channel is finished	00000038h
9.	System timer $\bar{O}1$	00000040h
10.	Trace	00000048h

5.1 Instruction Formats

The NMRC supports 32-bit instructions (see Fig. 5-1). There are three basic instruction types: control, address register modification and load-store-move instructions with the possibility of arithmetic operation setting at the same instruction. User defined instructions are also possible. All instructions are executed for one clock cycle.

Load and Store Instructions



Register to Register Move Instruction

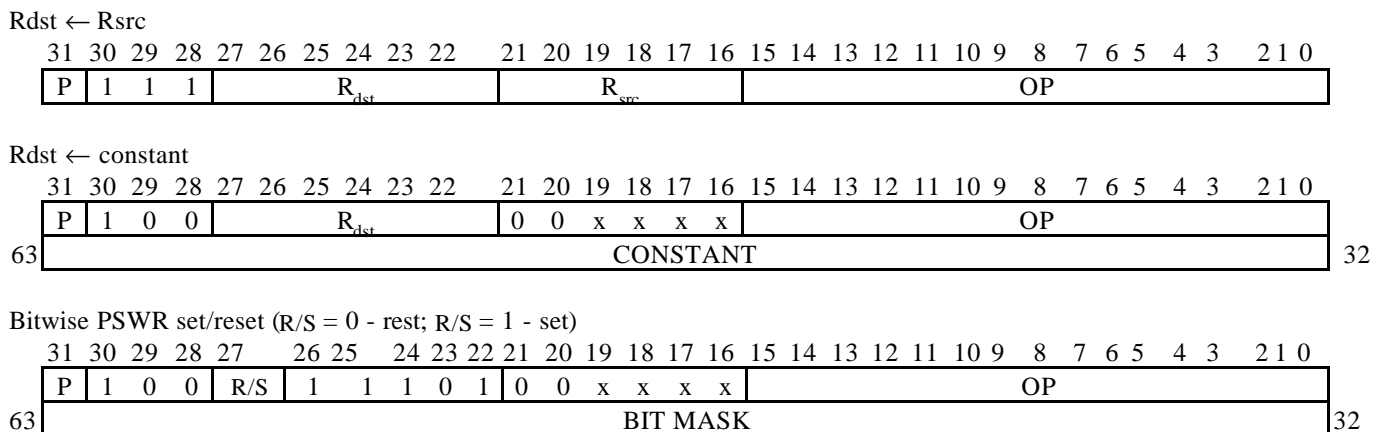


Fig. 5-1. Instruction Formats

Address Register Modification Instructions

$AR_j \leftarrow f_1(AR_i, GR_i)$

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
P	1	0	1	RM	1	AR _i	0	1	x	x	AR _j	OP																			

$AR_j \leftarrow f_1(AR_i, \text{Displacement})$

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
P	1	0	0	R _i	C	1	AR _i	0	1	x	x	AR _j	OP																		
63	CONSTANT DISPLACEMENT																									32					

No input/output and address modification operations

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
P	1	0	1	x	x	0	x	x	x	0	1	x	x	x	x	OP															

No input/output and address modification operations

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
P	1	0	0	x	x	0	x	x	x	0	1	x	x	x	x	OP															
63	CONSTANT DISPLACEMENT																									32					

Control Instructions

Jump/call to subroutine (J/C = 0 - jump; J/C = 1 - call to subroutine)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
P	0	0	0	PM	J/C	AR _i	1	0	Condition	OP																					

Jump/call to subroutine with displacement (J/C = 0 - jump; J/C = 1 - call to subroutine)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
P	1	0	0	PMC	J/C	AR _i	1	0	Condition	OP																					
63	ADDRESS (DISPLACEMENT)																									32					

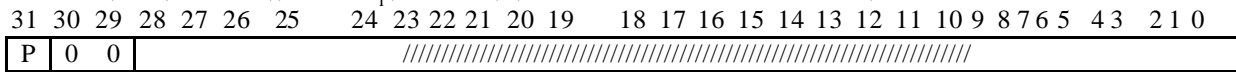
Return from subroutine/interrupt (S/I = 0 - return from subroutine; S/I = 1 - return from interrupt)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
P	0	0	0	S/I	0	1	1	1	1	1	1	1	Condition	OP																	

Fig. 5-1. Instruction Formats (Continued)

User Defined Instructions

AFIFO \leftrightarrow (fadr(ARi, GRi)); ARi \leftarrow f₁(ARi, GRi) (R/W = 0 - read; R/W = 1 - write)



Legend:

- AM - addressing mode fadr(ARi, GRi) with address register modification ARi \leftarrow f_m(ARi, GRi) (see Table 5-1)
- AMC- addressing mode using immediate constant fadr.(ARi, Displacement) with address register modification ARi \leftarrow f_m(ARi, Displacement) (see Table 5-2)
- RM - address register modification mode ARj \leftarrow f_m(ARi, GRi) (see Table 5-3)
- RMC - address register modification mode using immediate constant ARj \leftarrow f_m(ARi, Displacement) (see Table 5-4)
- PM - PC modification modes at branch instructions (see Table 5-5)
- PMC - PC modification modes at branch instructions using immediate constant (see Table 5-6)
- ARi - address register i (i = 0, ... , 7)
- ARj - address register j (j = 0, ... , 3 if 24th instruction bit is equal to 0, j = 4, ... , 7 if 24th instruction bit is equal to 1)
- Condition - condition code used for control instructions (see Table 5-7)
- Rsrc/dst - source/destination register for load, store and move instructions (see Table 5-8)
- OP - scalar arithmetic operation code (see chap. 5.7)
- P - parallel instruction flag (0 - disable execution in parallel for current instruction/ 1 - enable execution in parallel for current instruction)

Fig. 5-1. Instruction Formats (Continued)

5.2 Addressing Modes

Memory addressing modes for load and store instructions are defined at AM instruction field (see Table 5-1). Memory addressing modes for load and store instructions that use immediate constant are defined at AMC instruction field (see Table 5-2).

Table 5-1. Memory Addressing Modes for Load and Store Instructions

AM	$f_{adr.}(AR_i, GR_i)$	$f_m(AR_i, GR_i)$
0 0 0	GR_i	AR_i
0 0 1	AR_i+GR_i	AR_i+GR_i
0 1 0	GR_i	GR_i
0 1 1	Reserved	Reserved
1 0 0	AR_i	AR_i
1 0 1	AR_i	AR_i+GR_i
1 1 0	AR_i-a	AR_i-a
1 1 1	AR_i	AR_i+a

Table 5-2. Memory Addressing Modes for Load and Store Instructions with Immediate Constant

AMC	$f_{adr.}(AR_i, Displacement)$	$f_m(AR_i, Displacement)$
0 0	Address	AR_i
0 1	$AR_i+ Displacement$	$AR_i+ Displacement$
1 0	Address (Displacement)	Address (Displacement)
1 1	Reserved	Reserved

5.3 Address Register Modification Modes

Address register modification modes for corresponding instructions are defined by the respective RM instruction field (see Table 5-3). Address register modification modes with immediate constant for corresponding instructions are defined by the respective RMC instruction field (see Table 5-4).

Table 5-3. Address Register Modification Modes

RM	$f_m(AR_i, GR_i)$
0 0	AR_i
0 1	$AR_i + GR_i$
1 0	GR_i
1 1	Reserved

Table 5-4. Address Register Modification Modes with Immediate Constant

RMC	$f_i(AR_i, \text{Displacement})$
0 0	AR_i
0 1	$AR_i + \text{Displacement}$
1 0	Constant (Displacement)
1 1	Reserved

5.4 PC Modification Modes

PC modification modes for branch instructions are defined by the respective PM instruction field (see Table 5-5). PC modification modes for branch instructions with immediate constant are defined by the respective PMC instruction field (see Table 5-6).

Table 5-5. PC Modification Modes for Branch Instructions

PM	PC modification modes
0 0	PC:=ARi
0 1	PC:=ARi+GRi
1 0	PC:=GRi
1 1	PC:=PC+GRi

Table 5-6. PC Modification Modes for Branch Instructions with Immediate Constant

PMC	PC modification modes
0 0	PC:=ARi
0 1	PC:=ARi+ Displacement
1 0	PC:=Address (Displacement)
1 1	PC:=PC+ Displacement

5.5 Condition Codes and Flags

Control instructions can be unconditional or conditional depending on Condition instruction field. Decisions of conditional instruction executing are taken depending on the condition codes that are a combination of flags: N - negative, Z - zero, V - overflow, C - carry (see Table 5-7).

Table 5-7. NMRC Condition Codes

Condition	Flag combination
0 0 0 0	branch if C=0
0 0 0 1	branch if V=0
0 0 1 0	branch if N+Z=0
0 0 1 1	branch if N=0
0 1 0 0	branch if $(V \oplus N)+Z=0$
0 1 0 1	branch if $V \oplus N=0$
0 1 1 0	branch if Z=0
0 1 1 1	UNCONDITIONAL
1 0 0 0	branch if C=1
1 0 0 1	branch if V=1
1 0 1 0	branch if N+Z=1
1 0 1 1	branch if N=1
1 1 0 0	branch if $(V \oplus N)+Z=1$
1 1 0 1	branch if $V \oplus N=1$
1 1 1 0	branch if Z=1
1 1 1 1	NO BRANCH

5.6 Register Set

Load, store and move instructions field Rsrc/dst defines the register code (see Table 5-8). Depending on the code Rsrc/dst field defines 32-bit registers, 64-bit registers and 64-bit register pairs.

Table 5-8. Load, Store and Move Instructions Field Rsrc/dst

R _{src/dst}	Source register	Destination register	Width
0 0 0 0 0 0	AR0	AR0	32
0 0 0 0 0 1	AR1	AR1	32
0 0 0 0 1 0	AR2	AR2	32
0 0 0 0 1 1	AR3	AR3	32
0 0 0 1 0 0	AR4	AR4	32
0 0 0 1 0 1	AR5	AR5	32
0 0 0 1 1 0	AR6	AR6	32
0 0 0 1 1 1	AR7(SP)	AR7(SP)	32
0 0 1 0 0 0	OCA0	OCA0	32
0 0 1 0 0 1	ICA0	ICA0	32
0 0 1 0 1 0	OCA1	OCA1	32
0 0 1 0 1 1	ICA1	ICA1	32
0 0 1 1 0 0	T0	T0	32
0 0 1 1 0 1	--Reserved--	--Reserved--	32
0 0 1 1 1 0	--Reserved--	--Reserved--	32
0 0 1 1 1 1	PC	PC	32
0 1 0 0 0 0	GR0	GR0	32
0 1 0 0 0 1	GR1	GR1	32
0 1 0 0 1 0	GR2	GR2	32
0 1 0 0 1 1	GR3	GR3	32
0 1 0 1 0 0	GR4	GR4	32
0 1 0 1 0 1	GR5	GR5	32
0 1 0 1 1 0	GR6	GR6	32
0 1 0 1 1 1	GR7	GR7	32

Table 5-8. Load, Store and Move Instructions Field Rsrc/dst (Continued)

R _{src/dst}	Source register	Destination register	Width
0 1 1 0 0 0	OCC0	OCC0	32
0 1 1 0 0 1	ICC0	ICC0	32
0 1 1 0 1 0	OCC1	OCC1	32
0 1 1 0 1 1	ICC1	ICC1	32
0 1 1 1 0 0	T1	T1	32
0 1 1 1 0 1	--Reserved--	PSWRreset	32
0 1 1 1 1 0	INTR	--Reserved--	32
0 1 1 1 1 1	PSWR	PSWR	32
1 0 0 0 0 0	GR0, AR0	GR0, AR0	64(32+32)
1 0 0 0 0 1	GR1, AR1	GR1, AR1	64(32+32)
1 0 0 0 1 0	GR2, AR2	GR2, AR2	64(32+32)
1 0 0 0 1 1	GR3, AR3	GR3, AR3	64(32+32)
1 0 0 1 0 0	GR4, AR4	GR4, AR4	64(32+32)
1 0 0 1 0 1	GR5, AR5	GR5, AR5	64(32+32)
1 0 0 1 1 0	GR6, AR6	GR6, AR6	64(32+32)
1 0 0 1 1 1	GR7, AR7	GR7, AR7	64(32+32)
1 0 1 0 0 0	OCC0, OCA0	OCC0, OCA0	64(32+32)
1 0 1 0 0 1	ICC0, ICA0	ICC0, ICA0	64(32+32)
1 0 1 0 1 0	OCC1, OCA1	OCC1, OCA1	64(32+32)
1 0 1 0 1 1	ICC1, ICA1	ICC1, ICA1	64(32+32)
1 0 1 1 0 0	T1, T0	T1, T0	64(32+32)
1 0 1 1 0 1	DIR0	DOR0	64
1 0 1 1 1 0	DIR1	DOR1	64
1 0 1 1 1 1	PSWR, PC	PSWR, PC	64(32+32)
1 1 0 0 0 0	--Reserved--	--Reserved--	32
1 1 0 0 0 1	--Reserved--	--Reserved--	32
1 1 0 0 1 0	--Reserved--	--Reserved--	32
1 1 0 0 1 1	--Reserved--	--Reserved--	32
1 1 0 1 0 0	--Reserved--	--Reserved--	32
1 1 0 1 0 1	--Reserved--	--Reserved--	32
1 1 0 1 1 0	--Reserved--	--Reserved--	32
1 1 0 1 1 1	--Reserved--	--Reserved--	32
1 1 1 0 0 0	--Reserved--	--Reserved--	64
1 1 1 0 0 1	--Reserved--	--Reserved--	64
1 1 1 0 1 0	--Reserved--	--Reserved--	64
1 1 1 0 1 1	--Reserved--	--Reserved--	64
1 1 1 1 0 0	--Reserved--	--Reserved--	64
1 1 1 1 0 1	--Reserved--	PSWRset	32
1 1 1 1 1 0	--Reserved--	--Reserved--	32
1 1 1 1 1 1	--Reserved--	--Reserved--	32

5.7 Arithmetic and Logic Operations

In any instruction it is possible to set an arithmetic, logic or shift operation under general purpose registers GR0 - GR7 by OP instruction field (see Fig. 5-1). Instruction fields format is shown in Fig. 5-2.

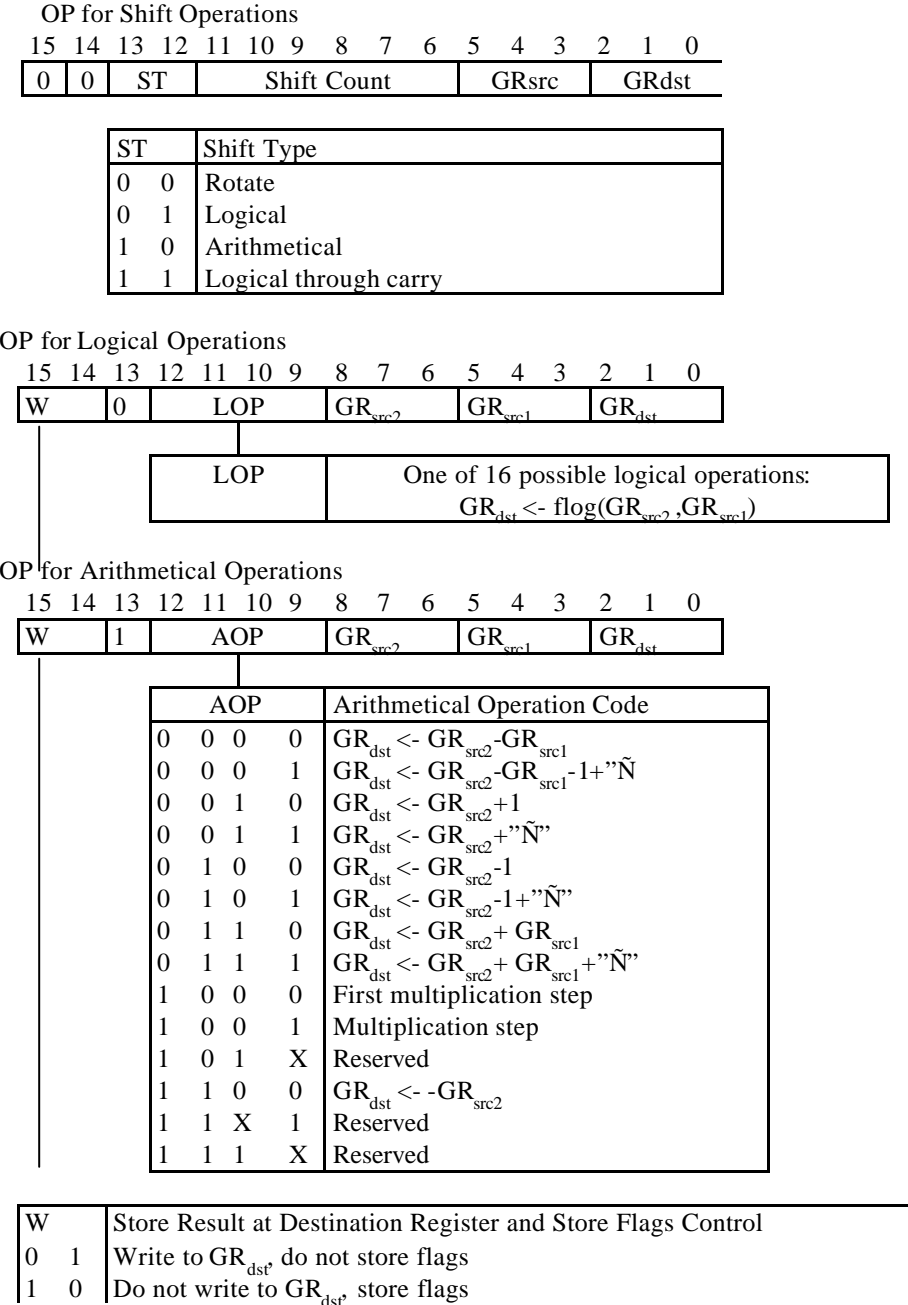


Fig. 5-2. OP Instruction Field Format

This page was intentionally left blank.



Research Centre Module
Box: 166, Moscow, 125190, Russia
Tel: +7 (095) 152-9335
Fax: +7 (095) 152-4661
E-Mail: info@module.ru
WWW: <http://www.module.ru>

©RC Module, 1998-2000

All rights reserved.

Neither the whole nor any part of the information contained in, or the product described in this overview may be adapted or reproduced in any form except with the prior written permission of the copyright holder.

RC Module reserves the right to make changes without further notice to product herein to improve reliability, function or design. RC Module shall not be liable for any loss or damage arising from the use of any information in this overview or any error or omission in such information, or any incorrect use of the product.

Printed in Russia Data of release: May 2000