BA22

32-bit RISC Processor

Programmer's Manual

CAST, Inc.

CONFIDENTIAL

1 Legal Notice

PLEASE READ THE FOLLOWING TERMS AND CONDITIONS CAREFULLY.

1.1 Disclaimer and "Terms of Use"

1.2 Trademarks

2 Table of Contents

2.1 Table Index

3 Conventions Used

b.mnemonic	Identifies a BA2 instruction. The length is the shortest that meets the requirements.
bt.mnemonic	Identifies a 16-bit wide BA2 instruction.
bn.mnemonic	Identifies a 24-bit wide BA2 instruction.
bg.mnemonic	Identifies a 32-bit wide BA2 instruction.
bw.mnemonic	Identifies a 48-bit wide BA2 instruction.
0x	Indicates a hexadecimal number.
rA	Instruction syntax used to identify a general purpose register.
REG[FIELD]	Syntax used to identify specific bit(s) of a general or special purpose register. FIELD can be a name of one bit or a group of bits or a numerical range constructed from two values separated by a colon.
X	In certain contexts, this indicates a "don't care".
N	In certain contexts, this indicates an undefined numerical value.
Implementation	An actual processor implementing the BA2 architecture.
Unit	Sometimes referred to as a co-processor. An implemented unit usually with some special registers and controlling instructions. It can be defined by the architecture or it may be custom.
Exception	A vectored transfer of control to supervisor software through an exception vector table. A way in which a processor can request operating system assistance (Reset, Bus Error, Data Page Fault, Instruction Page Fault, Tick Timer, Alignment, Interrupt, Illegal Instruction, Data TLB Miss, TLB Miss, Instruction, Range, Trap, System Call, Floating Point).
Privileged	An instruction (or register) that can only be executed (or accessed) when the processor is in supervisor mode (when SR[SM]=1).

Table 1: Conventions

4 Architecture Overview

The BA2 Architecture manual describes programming model of Beyond Semiconductor's BA2 family of processor cores. In itself BA2 is modern load/store RISC 32-bit architecture featuring industry highest code density in it's class. All this was achieved without compromise on performance, ease of use or scalability.

4.1 Features

The BA2 architecture includes the following principal features:

- A linear 32-bit logical address space with implementation-specific physical address space.
- Simple to use, orthogonal instruction set architecture with industry leading code density in it's class.
- Simple memory addressing modes.
- Configurable general purpose register file (from 16 - 32 GPR).

5 Addressing Modes and Operand Conventions

5.1 Memory Addressing Modes

The processor computes an effective address when executing a memory access instruction or branch instruction or when fetching the next sequential instruction. If the sum of the effective address and the operand length exceeds the maximum effective address in logical address space, the memory operand wraps around from the maximum effective address through effective address 0.

5.1.1 Register Indirect with Displacement

Load/store instructions using this address mode contain immediate value which can be either sign or zero extended and is added to the contents of a general-purpose register specified in the instruction.

Figure 1 shows how an effective address is computed when using register indirect with displacement addressing mode.

5.1.2 PC relative Addressing Mode

PC relative addressing mode is used by conditional and unconditional branch instructions. The displacement specified as an immediate value is sign-extended to 32 bits and added to PC.

Figure 2 shows how an effective address is computed when using PC relative addressing mode.

5.2 Memory Operand Conventions

The architecture defines an 8-bit byte, 16-bit halfword, a 32-bit word, and a 64-bit doubleword. It also defines IEEE-754 compliant 32-bit single precision float and 64-bit double precision float storage units.

Type of Data	Length in Bytes	Length in Bits
Byte	1	8
Halfword (or half)	2	16
Singleword (or word)	4	32
Doubleword (or double)	8	64
Single precision float	4	32
Double precision float	8	64

Table 2: Memory Operands and their sizes

5.2.1 Bit and Byte Ordering

Byte ordering defines how the bytes that make up halfwords, singlewords and doublewords are ordered in memory. To simplify BA2 implementations, the architecture implements Most Significant Byte (MSB) ordering - or big endian byte ordering by default. But implementations can support Least Significant Byte (LSB) ordering if defined during synthesis. Note that all devices, slaves and masters must be aware of the endianess used on the bus. The figures below illustrate the conventions for bit and byte numbering within various width storage units. These conventions hold for both integer and floating-point data, where the most significant byte of a floating-point value holds the sign and at least significant byte holds the start of the exponent.

bit 15 bit 8	bit 7 bit 0
MSB	LSB
Byte address 0	Byte address 1

Table 3: Default Bit and Byte Ordering in Halfwords

Table 3 shows how bits and bytes are ordered in a halfword.

bit 31 bit 24	bit 23 bit 16	bit 15 bit 8	bit 7 Bit 0
MSB			LSB
Byte address 0	Byte address 1	Byte address 2	Byte address 3

Table 4: Default Bit and Byte Ordering in Singlewords and Single Precision Floats

Table 4 shows how bits and bytes are ordered in a singleword.

bit 64 bit 56	bit 55 bit 48	bit 47 bit 40	bit 39 bit 32
MSB			
Byte address 0	Byte address 1	Byte address 2	Byte address 3

bit 31 bit 24	bit 23 bit 16	bit 15 bit 8	bit 7 bit 0
			LSB
Byte address 4	Byte address 5	Byte address 6	Byte address 7

Table 5: Default Bit and Byte Ordering in Doublewords, Double Precision Floats and all Vector Types

Table 6 shows how bits and bytes are ordered in a doubleword.

5.2.2 Aligned and Misaligned Accesses

A memory operand is naturally aligned if its address is an integral multiple of the operand length. Implementations might support accessing unaligned memory operands, but the default behavior is that accesses to unaligned operands result in an alignment exception. Check for information on alignment exception.

Operand	Length	addr[3:0] if aligned
Byte	8 bits	xxxx
Halfword (or half)	2 bytes	xxx0
Singleword (or word)	4 bytes	xx00
Doubleword (or double)	8 bytes	x000
Single precision float	4 bytes	xx00
Double precision float	8 bytes	x000
Vector of bytes	8 bytes	x000
Vector of halfwords	8 bytes	x000
Vector of singlewords	8 bytes	x000
Vector of single precision floats	8 bytes	x000

Table 6: Memory Operand Alignment

BA2 instructions are 3, 4, 6 bytes long and are not word-aligned.

6 Application Binary Interface v3 (ABIv3)

6.1 Data Representation

6.1.1 Fundamental Types

Scalar types in the ISO/ANSI C language are based on memory operands definitions from the chapter entitled Addressing Modes and Operand Conventions. Similar relations between architecture and language types can be used for any other language.

TYPE	C TYPE	SIZEOF	ALIGNMENT (BYTES)	BA2 EQUIVALENT
Integral	Char Signed char	1	1	Signed byte
Integral	Unsigned char	1	1	Unsigned byte
Integral	Short Signed short	2	2	Signed halfword
Integral	Unsigned short	2	2	Unsigned halfword
Integral	Int Signed int Long Signed long Enum	4	4	Signed singleword
Integral	Unsigned int	4	4	Unsigned singleword
Integral	Long long Signed long long	8	8	Signed doubleword
Integral	Unsigned long long	8	8	Unsigned doubleword
Pointer	Any-type * Any-type (*) ()	4	4	Unsigned singleword
Floating-point	Float	4	4	Single precision float
Floating-point	Double	8	8	Double precision float

Table 7: Scalar Types

A null pointer of any type must be zero. All floating-point types are IEEE-754 compliant.

The BA2 programming model introduces a set of fundamental vector data types, as described by Table 16-2. For vector assignments both side of assignment must be of the same vector type.

VECTOR TYPE	SIZEOF	ALIGNMENT (BYTES)	BA2 EQUIVALENT
Vector char Vector signed char	8	8	Vector of signed bytes
Vector unsigned char	8	8	Vector of unsigned bytes
Vector short Vector signed short	8	8	Vector of signed halfwords
Vector int Vector signed int Vector long Vector signed long	8	8	Vector of signed singlewords
Vector unsigned int	8	8	Vector of unsigned singlewords
Vector float	8	8	Vector of single-precisions

Table 8: Vector Types

6.1.2 Aggregates and Unions

Aggregates (structures and arrays) and unions assume the alignment of their most strictly aligned element.

- An array uses the alignment of its elements.
- Structures and unions can require padding to meet alignment restrictions. Each element is assigned to the lowest aligned address.

```
struct { char C; };
struct { char C; char D; short S; long N; };
struct { char C; double D; short S; };
```

6.1.3 Bit-fields

C structure and union definitions can have elements defined by a specified number of bits. Table 9 describes valid bit-field types and their ranges.

Bit-field	Type Width w [bits]	Range
Signed char	1 to 8	-2w-1 to 2w-1-1
Char	1 to 8	0 to 2w-1
Unsigned char	1 to 8	0 to 2w-1
Signed short	1 to 16	-2w-1 to 2w-1-1
Short	1 to 16	0 to 2w-1
Unsigned short	1 to 16	0 to 2w-1
Signed int	1 to 32	-2w-1 to 2w-1-1
Int	1 to 32	0 to 2w-1
Enum	1 to 32	0 to 2w-1
Unsigned int	1 to 32	0 to 2w-1
Signed long	1 to 32	-2w-1 to 2w-1-1
Long	1 to 32	0 to 2w-1
Unsigned long	1 to 32	0 to 2w-1

Table 9: Bit-Field Types and Ranges

Bit-fields follow the same alignment rules as aggregates and unions, with the following additions:

- Bit-fields are allocated from most to least significant (from left to right).
- A bit-field must entirely reside in a storage unit appropriate for its declared type.
- Bit-fields may share a storage unit with other struct/union elements, including elements that are not bit-fields. Struct elements occupy different parts of the storage unit.
- Unnamed bit-fields' types do not affect the alignment of a structure or union.

S(9)	J (9)	Pad (6)	C (8)
T(9)	Pad (7)	U (9)	Pad (7)
D(8)	Pad (24)	Pad (24)	Pad (24)

Table 10: Storage unit sharing abd alignment padding, sizeof is 12

6.2 Function Calling Sequence

This section describes the standard function calling sequence, including stack frame layout, register usage, parameter passing, and so on. The standard calling sequence requirements apply only to global functions, however it is recommended that all functions use the standard calling sequence.

6.3 Register Usage

The BA2 defines 32 general-purpose registers. These registers are 32 bits wide in 32-bit implementations and 64 bits wide in 64-bit implementations.

Register	Preserved across function calls	Usage
R31	NO*	Temporary register*
R30	NO	Temporary register
R29	NO	Temporary register
R28	NO	Temporary register
R27	NO	Temporary register
R26	NO	Temporary register
R25	NO	Temporary register
R24	NO	Temporary register
R23	NO	Temporary register
R22	YES	Callee-saved register
R21	YES	Callee-saved register
R20	YES	Callee-saved register
R19	YES	Callee-saved register
R18	YES	Callee-saved register
R17	YES	Callee-saved register
R16	YES	Callee-saved register
R15	YES	Callee-saved register
R14	YES	Callee-saved register
R13	YES	Callee-saved register
R12	YES	Callee-saved register
R11	YES	Callee-saved register
R10	YES	FP - Frame pointer
R9	YES	LR - Link address register
R8	NO	Function argument word 5
R7	NO	Function argument word 4
R6	NO	Function argument word 3
R5	NO	Function argument word 2
R4	NO	Function argument word 1 RVH - Return value high
R3	NO	Function argument word 0 RV - Return value

R2	NO	GP - Global pointer register
R1	YES	SP - Stack pointer
R0	-	Fixed to zero

Table 11: Registers

Some registers have assigned roles:

R0 - [Zero] - Always **fixed to zero**. Even if it is writable in some embedded implementations, the software shouldn't modify it.

R1 - [SP] - The **stack pointer** holds the limit of the current stack frame. The stack contents below the stack pointer are undefined. Exception to this rule is usage of red zone space in leaf functions. Stack pointer must be double word aligned at all times.

R2 - [GP] - The **global pointer** register points into the small data area. It is intended for data, addressed with short offsets relative to the global pointer register. If short addressing is not used, GP register can be used as temporary register, not preserved across function calls.

R3 through R8 - **General-purpose parameters** use up to 6 general-purpose registers. Parameters beyond the sixth parameter appear on the stack.

R3 - [RV] - **Return value** of the function. For void functions a value is not defined. For functions returning a union or structure, a pointer to the result is placed into return value register.

R4 - [RVH] - **Return value high** of the function. For functions returning 32-bit values this register can be considered temporary register.

R9 - [LR] - **Link address** is the location of the function call instruction and is used to calculate where program execution should return after function completion.

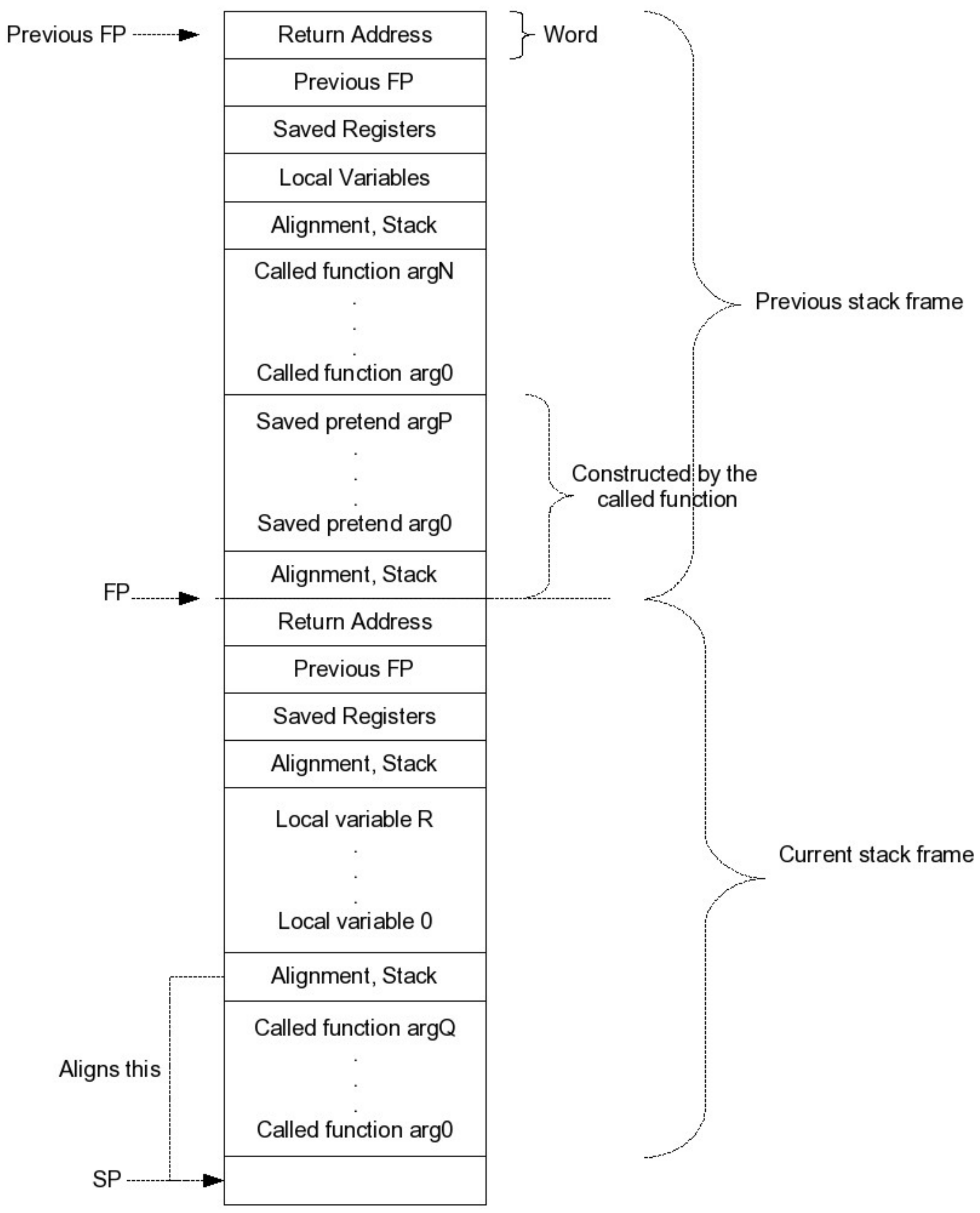
R10 - [FP] - The **frame pointer** holds the address of the previous stack frame. Incoming function parameters reside in the previous stack frame and can be accessed at positive offsets from FP.

R31 – [PC] – It is generally a temporary register. Depending on special CPU configuration it is used as **Program Counter** storage. The PC address is rounded down to the nearest 32-bit boundary.

6.3.1 The Stack Frame

In addition to registers, each function has a frame on the run-time stack. This stack grows downward from high addresses.

The stack pointer always points to the end of the latest allocated stack frame. The first 512 bytes below the stack frame are reserved for leaf functions that do not need to modify their stack pointer. Exception handlers must guarantee that they will not use this area. Functions that have a 64-bit argument split between r8 and stack, internally decrement SP by 4 bytes to reconstruct passed 64-bit value on the stack. Leaf functions do not need to create a stack slot to save their return address, and functions that omit frame pointer do not need to create a stack slot to save previous FP value.



Picture 1: Stack frame layout

6.3.2 Parameter Passing

Functions receive their first 6 argument words in general-purpose function argument registers r3-r8. If there are more than six argument words, the remaining argument words are passed on the stack. Small structure and union arguments (not more than 64 bits in total) are passed in argument registers, other structure and union arguments are passed as pointers to memory locations that contain the arguments itself. All 64-bit arguments in a 32-bit system are passed using a pair of registers. 64-bit argument need not to be aligned on 32-bit boundaries.

For example long long arg1, long arg2, long long arg3, long long arg4 are to be passed in the following way: arg1 in r3 and r4, arg2 in r5, arg3 in r6 and r7. Argument 4 has to be split between r8 and stack, and the receiving function should reconstruct the passed value.

6.3.3 Functions Returning Scalars or Nor Value

A function that returns an integral, floating point, vector or pointer value, puts its result in the general-purpose RV register pair. Void functions put no particular value in the RV register pair.

6.3.4 Functions Returning Structures or Unions

A function that returns a small structure or union (not more than 64 bits in total) places returned value in the general purpose RV register pair. Other structures and unions are returned in the memory, pointed by the "invisible" first function argument.

6.3.5 Functions Returning Complex Values

A function that returns a complex value (not more than 64 bits in total) places returned value in the general purpose RV register pair. Complex values more than 64 bits wide are returned in the memory, pointed by the "invisible" first function argument.

7 Register Set

7.1 Features

The BA2 register set includes the following principal features:

- User selectable number up to 32 general-purpose registers BA2 implementations optimized for use in FPGAs and ASICs in embedded and similar environments may implement only subset of the possible thirty-two registers.
- All other registers are special-purpose registers defined for each unit separately and accessible through the b.mtspr/b.mfspr instructions.

7.2 Overview

A BA2 processor includes several types of registers: user level general-purpose and special-purpose registers, supervisor level special-purpose registers and unit-dependent registers. User level general-purpose and special-purpose registers are accessible both in user mode and supervisor mode of operation. Supervisor level special-purpose registers are accessible only in supervisor mode of operation (SR[SM]=1). Unit dependent registers are usually only accessible in supervisor mode but there can be exceptions to this rule. Accessibility for architecture-defined units is defined in this manual. Accessibility for custom units not covered by this manual is defined in the appropriate implementation-specific manuals.

7.3 Special-Purpose Registers

The special-purpose registers of all units are grouped into thirty-two groups. Each group can have different register address decoding depending on the maximum theoretical number of registers in that particular group. A group can contain registers from several different units or processes. The SR[SM] bit is also used in register address decoding, as some registers are accessible only in supervisor mode. The b.mtspr and b.mfspr instructions are used for reading and writing registers.

GROUP	UNIT DESCRIPTION
0	System Control and Status registers
1	Data MMU (in the case of a single unified MMU, groups 1 and 2 decode into a single set of registers)
2	Instruction MMU (in the case of a single unified MMU, groups 1 and 2 decode into a single set of registers)
3	Data Cache (in the case of a single unified cache, groups 3 and 4 decode into a single set of registers)
4	Instruction Cache (in the case of a single unified cache, groups 3 and 4 decode into a single set of registers)
5	MAC unit

6	Debug unit
7	Performance counters unit
8	Power Management
9	Programmable Interrupt Controller
10	Tick Timer
12-23	Reserved for future use
24-31	Custom units

Table 12: Groups of SPRs

A BA2 processor implementation is required to implement at least the special purpose registers from group 0. All other groups are optional, and registers from these groups are implemented only if the implementation has the corresponding unit. Which units are actually implemented may be determined by reading the UPR register from group 0. A 16-bit SPR address is made of 5-bit group index (bits 15-11) and 11-bit register index (bits 10-0).

Grp	Reg	Reg Name	USER MODE	SUPV MODE	Description
0	0	VR	-	R	Version register
0	1	UPR	-	R	Unit Present register
0	2	CPUCFGR	-	R	CPU Configuration register
0	3	DMMUCFGR	-	R	Data MMU Configuration register
0	4	IMMUCFGR	-	R	Instruction MMU Configuration register
0	5	DCCFGR	-	R	Data Cache Configuration register
0	6	ICCFGR	-	R	Instruction Cache Configuration register
0	7	DCFGR	-	R	Debug Configuration register

0	8	PCCFGR	-	R	Performance Counters Configuration register
0	9	CPUID	-	R	CPU Identification register
0	16	NPC	-	R/W	PC mapped to SPR space (next PC)
0	17	SR	-	R/W	Supervision register
0	18	PPC	-	R/W	PC mapped to SPR space (previous PC)
0	19	UR	W	R/W	User register
0	32-47	EPCR	-	R/W	Exception PC register
0	48-63	EEAR	-	R/W	Exception EA register
0	64-79	ESR	-	R/W	Exception SR register
0	0x400 + (x)	GPR(x)	-	-	GPRs mapped to SPR space (read/write)
1	2	DTLBEIR	-	W	Data TLB Entry invalidate register
1	4	DTLBMR	-	W	Data TLB Match register
1	6	DTLBTR	-	W	Data TLB Translate register
2	2	ITLBEIR	-	W	Instruction TLB Entry invalidate register
2	4	ITLBMR	-	W	Instruction TLB Translation register
2	6	ITLBTR	-	W	Instruction TLB Translate register

3	2	DCBFR	-	W	DC Block Flush register
3	3	DCBIR	-	W	DC Block Invalidate register
4	2	ICBIR	-	W	IC Block Invalidate register
5	1	MACLO	R/W	R/W	MAC Low
5	2	MACHI	R/W	R/W	MAC High
6	0-7	DVR0-DVR7	-	R/W	Debug Value registers
6	8-15	DCR0-DCR7	-	R/W	Debug Control registers
6	16	DMR1	-	R/W	Debug Mode register 1
6	20	DSR	-	R/W	Debug Stop register
6	21	DRR	-	R/W	Debug Reason register
8	0	PMEVNT	-	R/W	Power Management Event Control register
8	2	PMUNITPD	-	R/W	Power Management Unit(s) Power Down register
9	0	PICMR	-	R/W	PIC Mask register
9	2	PICSR	-	R/W	PIC Status register
10	0	TTMR	-	R/W	Tick Timer Mode register
10	1	TTCR	R*	R/W	Tick Timer Count register

Table 13: List of All Special-Purpose Registers

7.4 General-Purpose Registers (GPRs)

The thirty-two general-purpose registers are labeled R0-R31 and are 32 bits wide in 32-bit implementations and 64 bits wide in 64-bit implementations. They hold scalar integer data, floating-point data, vectors or memory pointers.

Register					r31	r30
Register	R29	R28	r27	r26	r25	r24
Register	R23	R22	r21	r20	r19	r18
Register	R17	R16	r15	r14	r13	r12
Register	R11	r10	r9	r8	r7	r6
Register	R5	r4	r3	r2	r1	r0

Table 14: General-Purpose Registers

Table 17 contains a list of general-purpose registers. The GPRs may be accessed as both source and destination registers. See chapter Application Binary Interface for information on floating-point data types.

R0 is used as a constant zero. Whether or not R0 is actually hardwired to zero is implementation dependent. **R0 should never be used as a destination register.** Functions of other registers are explained in chapter Application Binary Interface. The reset exception handler is responsible for initializing GPRs to zero if that is necessary.

7.5 Support for Custom Number of GPRs

Programs may be compiled with less than thirty-two registers. Unused registers are disabled (set as *fixed* registers) when compiling code. Such code is also executable on normal implementations with thirty-two registers but not vice versa. This feature is quite useful since users are expected to move from less powerful BA2 implementations with less than thirty-two registers to more powerful thirty-two register BA2 implementations. If configuration registers are implemented, CPUCFGR[CGF] indicates whether implementation has complete thirty-two general-purpose registers or less than thirty-two registers.

7.6 Supervision Register (SR)

The supervision register is a 32-bit special-purpose supervisor-level register accessible with the b.mtspr/b.mfspr instructions in supervisor mode only. The SR value defines the state of the processor.

Bit	31 ... 25	24
Identifier	Reserved	FPEE_FDZ
Reset	0	0
R/W	Read Only	R/W

Bit	23	22	21	20	19	18	17	16
Identifier	FPEE_FIN	FPEE_FIV	FPEE_FIX	FPEE_FZ	FPEE_FQN	FPEE_FSN	FPEE_FUN	FPEE_FOV
Reset	0	0	0	0	0	0	0	0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

Bit	15	14	13	12	11	10	9	8
Identifier	FO	EPH	ST	TED	OV	CY	F	SBE
Reset	1	0	0	0	0	0	0	0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

Bit	7	6	5	4	3	2	1	0
Identifier	Reserved	IME	DME	ICE	DCE	IEE	TEE	SM
Reset	0	0	0	0	0	0	0	1
R/W	Read Only	R/W	R/W	R/W	R/W	R/W	R/W	R/W

SM	Supervisor Mode
0	Processor is in User Mode
1	Processor is in Supervisor Mode
TEE	Tick Timer Exception Enabled
0	Tick Timer Exceptions are not recognized
1	Tick Timer Exceptions are recognized
IEE	Interrupt Exception Enabled
0	Interrupts are not recognized
1	Interrupts are recognized

DCE	Data Cache Enable
0	Data Cache is not enabled
1	Data Cache is enabled
ICE	Instruction Cache Enable
0	Instruction Cache is not enabled
1	Instruction Cache is enabled
DME	Data MMU Enable
0	Data MMU is not enabled
1	Data MMU is enabled
IME	Instruction MMU Enable
0	Instruction MMU is not enabled
1	Instruction MMU is enabled
SBE	Store Buffer Enable
0	Store Buffer is disabled
1	Store Buffer is enabled
F	Flag
0	Conditional branch flag was cleared by sfXX instructions
1	Conditional branch flag was set by sfXX instructions
CY	Carry flag
0	No carry out produced by last arithmetic operation
1	Carry out was produced by last arithmetic operation
OV	Overflow flag
0	No overflow occurred during last arithmetic operation
1	Overflow occurred during last arithmetic operation
TED	Trap Exception Disabled
0	Trap exception is enabled
1	Trap exception is disabled
ST	Single-step Trace
0	Single-step trace disabled
1	Every executed instruction causes trap exception
EPH	Exception Prefix High

0	Exceptions vectors are located in memory area starting at 0x0
1	Exception vectors are located in memory area starting at 0xF0000000
FO	Fixed One
	This bit is always set
FPEE_FOV	FP OVerflow Exception Enable
0	Exception is disabled
1	Exception is enabled
FPEE_FUN	FP UNderflow Exception Enable
0	Exception is disabled
1	Exception is enabled
FPEE_FSN	FP SNAN Exception Enable
0	Exception is disabled
1	Exception is enabled
FPEE_FQN	FP QNAN Exception Enable
0	Exception is disabled
1	Exception is enabled
FPEE_FZ	FP Zero Exception Enable
0	Exception is disabled
1	Exception is enabled
FPEE_FIX	FP IneXact Exception Enable
0	Exception is disabled
1	Exception is enabled
FPEE_FIV	FP InValid Exception Enable
0	Exception is disabled
1	Exception is enabled
FPEE_FIN	FP INfinity Exception Enable
0	Exception is disabled
1	Exception is enabled
FPEE_FDZ	FP Divide by Zero Exception Enable
0	Exception is disabled
1	Exception is enabled

Table 15: SR Field Descriptions

7.7 Exception Program Counter Registers (EPCR)

The Exception Program Counter register is special-purpose supervisor-level register accessible with the b.mtspr and b.mfspr instructions in supervisor mode. The EPCR is 32-bit wide register in 32-bit implementations and can be wider than 32 bits in 64-bit implementations. After an exception, the EPCR is set to the program counter address (PC) of the instruction that was interrupted by the exception. The EPCR must be saved by the exception handler routine before exception recognition is re-enabled in the SR.

Bit	31-0
Identifier	EPC
Reset	0
R/W	R/W

EPC	Exception Program Counter Address

Table 16: EPCR Field Descriptions

7.8 Exception Effective Address Registers (EEAR)

The Exception Effective Address register is special-purpose supervisor-level register accessible with the b.mtspr/b.mfspr instructions in supervisor mode. The EEAR is 32-bit wide register in 32-bit implementations and can be wider than 32 bits in 64-bit implementations. After an exception, the EEAR is set to the effective address (EA) generated by the faulting instruction. The EEAR must be saved by the exception handler routine before exception recognition is re-enabled in the SR.

Bit	31-0
Identifier	EEA
Reset	0
R/W	R/W

EEA	Exception Effective Address

Table 17: EEAR Field Descriptions

7.9 Exception Supervision Register (ESR)

The Exception Supervision register is special-purpose supervisor-level register accessible with b.mtspr/b.mfspr instructions in supervisor mode. The ESR is 32 bits wide register in 32-bit implementations and can be wider than 32 bits in 64-bit implementations. After an exception, the Supervision register (SR) is copied into the ESR. The ESR must be saved by the exception handler routine before exception recognition is re-enabled in the SR.

Bit	31-0
Identifier	ESR

Reset	0
R/W	R/W

ESR	Exception SR

Table 18: ESR Field Descriptions

7.10 User Register (UR)

User register is a 32-bit special-purpose register accessible with the b.mtspr/b.mfspr instructions. It controls floating point rounding modes and provides floating point status flags. Status flags are updated after every floating point instruction is completed and can serve to determine what caused the floating point exception. If floating point exception is enabled then status flags have to be cleared in floating point exception handler. Status flags are cleared by writing 0 to all status bits.

Bit	31	30 ... 27	26 25	24
Identifier	DPFW	Reserved	FRM	FDZ
Reset	0	0	0	0
R/W	R/W	Read Only	R/W	R/W

Bit	23	22	21	20	19	18	17
Identifier	FIN	FIV	FIX	FZ	FQN	FSN	FUN
Reset	0	0	0	0	0	0	0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

Bit	16	15 - 13	12	11	10
Identifier	FOV	Reserved	MACOVL	MACOVH	MACCY
Reset	0	0	0	0	0
R/W	R/W	Read Only	R/W	R/W	R/W

Bit	9 ... 0
Identifier	Reserved
Reset	0
R/W	Read Only

MACCY	MAC Set Carry
0	No Carry
1	Carry occurred
MACOVH	MAC Set Overflow High
0	No overflow
1	Result overflowed
MACOVL	MAC Set Overflow Low
0	No overflow
1	Result overflowed
FOV	OVerflow Flag
0	No overflow
1	Result overflowed
FUN	UNderflow Flag
0	No underflow
1	Result underflowed
FSN	SNAN Flag
0	Result not SNAN
1	Result SNAN
FQN	QNAN Flag
0	Result not QNAN
1	Result QNAN
FZ	Zero Flag
0	Result not zero
1	Result zero
FIX	IneXact Flag
0	Result precise
1	Result inexact
FIV	InValid Flag
0	Result valid
1	Result invalid
FIN	INfinity Flag
0	Result finite

1	Result infinite
FDZ	Divide by Zero Flag
0	Proper divide
1	Divide by zero
FRM	Rounding Mode
00	Round to nearest
01	Round to zero
10	Round to infinity+
11	Round to infinity-
DPFW	Page Fault
0	Data page fault was a read
1	Data page fault was a write

Table 19: UR Field Description

7.11 CPU Identification Register (CPUID)

CPU Identification register is a 32-bit special-purpose register accessible with the b.mfspr instructions. It enables the identification of the CPU core in multiprocessor environment.

Bit	31-0
Identifier	CPUID
Reset	0
R/W	R

CPUID	CPU Identification in a multiprocessor environment

Table 20: CPUID Field Descriptions

8 Exception Model

This chapter describes the various exception types and their handling.

8.1 Introduction

The exception mechanism allows the processor to change to supervisor state as a result of external signals, errors, or unusual conditions arising in the execution of instructions. When exceptions occur, information about the state of the processor is saved to certain registers and the processor begins execution at the address predetermined for each exception. Processing of exceptions begins in supervisor mode. The architecture requires that all exceptions be handled in strict order with respect to the instruction stream. When an instruction-caused exception is recognized, any unexecuted instructions that appear earlier in the instruction stream are required to complete before the exception is taken. Exceptions can occur while an exception handler routine is executing, and multiple exceptions can become nested.

8.2 Exception Classes

All exceptions can be described as precise or imprecise and either synchronous or asynchronous. Synchronous exceptions are caused by instructions and asynchronous exceptions are caused by events external to the processor.

Type	Exception
Asynchronous/nonmaskable	Bus Error, Reset
Asynchronous/maskable	External Interrupt, Tick Timer
Synchronous/precise	Instruction-caused exceptions
Synchronous/imprecise	None

Table 21: Exception Classes

Exception Type	Vector Offset	Causal Conditions
Reset	0x100	Caused by software or hardware reset.
Bus Error	0x200	The causes are implementation-specific, but typically they are related to bus errors and attempts to access invalid physical address.
Data Page Fault	0x300	Page protection violation for load/store operations.
Instruction Page Fault	0x400	Page protection violation for instruction fetch.
Tick Timer	0x500	Tick timer interrupt asserted.

Alignment	0x600	Load/store access to naturally not aligned location.
Illegal Instruction	0x700	Illegal instruction in the instruction stream.
External Interrupt	0x800	External interrupt asserted.
D-TLB Miss	0x900	No matching entry in DTLB (DTLB miss).
I-TLB Miss	0xA00	No matching entry in ITLB (ITLB miss).
Range	0xB00	If programmed in the SR, the setting of certain flags, like SR[OV], causes a range exception. On BA2 implementations with less than 32 GPRs when accessing unimplemented architectural GPRs. On all implementations if SR[CID] had to go out of range in order to process next exception.
System Call	0xC00	System call initiated by software.
Floating Point	0xD00	Caused by floating point instructions when status flags are set by FPU and corresponding FPU Exception bit is set in SR.
Trap	0xE00	Caused by the b.trap instruction or by debug unit.

Table 22: Exception Types and Causal Conditions

Whenever an exception occurs, current PC is saved to current EPCR and new PC is set with the vector address according to Table 22.

8.3 Exception Processing

Whenever an exception occurs, the processor architectural state needed to continue execution after exception is processed, is stored into so called exception registers. These are:

- EPCR (Exception PC register). At the time exception occurs, EPCR will be assigned a PC value that should be used to continue execution after exception is handled. It points either to the next instruction in the execution flow or it may also point to instruction that was not retired at the time of exception and thus needs to be repeated. Examples are function epilogue (b.entri) and function prologue (b.reti, b.rtnei) instructions.
- ESR (Exception SR register). It holds the last value of SR register before the exception happened.
- EEAR (Exception EA register). Effective address of the data access that caused the exception is

placed into EEAR register. This register holds defined value only if one of the following exceptions occurred: Bus Error, IMMU page fault, DMMU page fault, Alignment, I-TLB miss, D-TLB miss.

- JPC (Jump Program Counter). This register is always pointed to the PC of the last jumping instruction before the exception. It can be very useful software debugging aid.

When exception handler returns via b.rfe (return from exception) instruction processor state in SR register will be set with content of ESR and EPCR value will be placed into processor's PC. EEAR and JPC registers are only registers holding additional info necessary or useful for exception handling and software debugging.

Exception	Priority	EPCR	EEAR
Reset	1	-	-
Bus Error	4 (insn) 9 (data)	Address of instruction that caused exception	Load/store/fetch virtual EA
Data Page Fault	8	Address of instruction that caused exception	Load/store virtual EA
Instruction Page Fault	3	Address of instruction that caused exception	Instruction fetch virtual EA
Tick Timer	12	Address of next not executed instruction	-
Alignment	6	Address of instruction that caused exception	Load/store virtual EA
Illegal Instruction	5	Address of instruction that caused exception	Instruction fetch virtual EA
External Interrupt	12	Address of next not executed instruction	-
D-TLB Miss	7	Address of instruction that caused exception	Load/store virtual EA
I-TLB Miss	2	Address of instruction that caused exception	Instruction fetch virtual EA
Range	10	Address of instruction that caused exception	-
System Call	7	Address of next not executed instruction	-
Floating Point	11	Address of instruction that caused exception	-
Trap	7	Address of instruction that caused exception	-

Table 23: Values of EPCR and EEAR after Exception

All registers that will be modified by exception handler routine must first be saved. All exceptions set a new SR where both MMUs are disabled (address translation disabled), supervisor mode is turned on, and tick timer exceptions and interrupts are disabled. (SR[DME]=0, SR[IME]=0, SR[SM]=1, SR[IEE]=0 and

SR[TEE]=0). When enough machine state information has been saved by the exception handler, SR[TTE] and SR[IEE] can be re-enabled so that tick timer and external interrupts are not blocked. When returning from an exception handler with b.rfe, SR and PC are restored but general-purpose registers previously saved by exception handler need to be restored as well.

9 Memory Model

This chapter describes the BA2 weakly ordered memory model.

9.1 Memory

Memory is byte-addressed with halfword accesses aligned on 2-byte boundaries, singleword accesses aligned on 4-byte boundaries, and doubleword accesses aligned on 8-byte boundaries.

9.2 Memory Access Ordering

The BA2 architecture specifies a weakly ordered memory model for uniprocessor and shared memory multiprocessor systems. This model has the advantage of a higher-performance memory system but places the responsibility for strict access ordering on the programmer. The order in which the processor performs memory access, the order in which those accesses complete in memory, and the order in which those accesses are viewed by another processor may all be different. Two means of enforcing memory access ordering are provided to allow programs in uniprocessor and multiprocessor system to share memory. An BA2 processor implementation may also implement a more restrictive, strongly ordered memory model. Programs written for the weakly ordered memory model will automatically work on processors with strongly ordered memory model.

9.2.1 Memory Synchronize Instruction

The **b.msync** instruction permits the program to control the order in which load and store operations are performed. This synchronization is accomplished by requiring programs to indicate explicitly in the instruction stream, by inserting a memory sync instruction, that synchronization is required. The memory sync instruction ensures that all memory accesses initiated by a program have been performed before the next instruction is executed. BA2 processor implementations, that implement the strongly-ordered memory model instead of the weakly-ordered one, can execute memory synchronization instruction as a no-operation instruction.

9.2.2 Pages Designated as Weakly-Ordered-Memory

When a memory page is designated as a Weakly-Ordered-Memory (WOM) page, instructions and data can be accessed out-of-order and with prefetching. When a page is designated as not WOM, instruction fetches and load/store operations are performed in-order without any prefetching. BA2 scalar processor implementations, that implement strongly-ordered memory model instead of the weakly-ordered one and perform load and store operations in-order, are not required to implement the WOM bit in the MMU.

10 Debug Unit (Optional)

This chapter describes the BA2 debug facility. The debug unit assists software developers in debugging their systems. It provides support for watchpoints, breakpoints and program-flow control registers. Watchpoints and breakpoints may trigger events causing the debug interface to stall the processor or when a debug interface is not used, invoke the resident debugger software.

10.1 Features

The BA2 architecture defines eight sets of debug registers. Additional debug register sets can be defined by the implementation itself. The debug unit is optional and the presence of an implementation is indicated by the UPR[DUP] bit.

- Optional implementation
- Eight architecture defined sets of debug value/compare registers
- Match signed/unsigned conditions on instruction fetch EA, load/store EA and load/store data
- Match conditions on instruction fetch address can generate a breakpoint (trap exception). The program is interupted before the instruction matching conditions is comitted.
- Match conditions on load/store EA or load/store data or combining match conditions for complex watchpoints (trap exception). The program is interupted/stalled after the instruction matching the condition is comited

10.2 Debug Value Registers (DVR0-DVR7)

The debug value registers are 32-bit special-purpose supervisor-level registers accessible with the b.mtspr/b.mfspr instructions in supervisor mode. The DVRs are programmed with the comparison addresses or data by the resident debug software or by the development interface. Their value is compared to the fetch or load/store EA or to the load/store data according to the corresponding DCR. Based on the settings of the corresponding DCR a match condition is generated.

Bit	31-0
Identifier	VALUE
Reset	0
R/W	R/W

VALUE	Watchpoint/Breakpoint Address/Data

Table 24: DVR Field Descriptions

10.3 Debug Control Registers (DCR0-DCR7)

The debug control registers are 32-bit special-purpose supervisor-level registers accessible with the b.mtspr/b.mfspr instructions in supervisor mode. The DCRs are programmed with the watchpoint settings that define how DVRs are compared to the instruction fetch or load/store EA or to the load/store data.

Bit	31 ... 8	7 6 5	4	3 2	0
Identifier	Reserved	CT	SC	CC	DP
Reset	X	0	0	0	0
R/W	R	R/W	R/W	R/W	R

DP	DVR/DCR Present
0	Corresponding DVR/DCR pair is not present
1	Corresponding DVR/DCR pair is present
CC	Compare Condition
000	Masked
001	Equal
010	Less than
011	Less than or equal
100	Greater than
101	Greater than or equal
110	Not equal
111	Reserved
SC	Signed Comparison
0	Compare using unsigned integers
1	Compare using signed integers
CT	Compare To
000	Comparison disabled
001	Instruction fetch EA
010	Load EA
011	Store EA
100	Load data
101	Store data
110	Load/Store EA

111	Load/Store data

Table 25: DCR Field Descriptions

10.4 Debug Mode Register 1 (DMR1)

The debug mode register 1 is a 32-bit special-purpose supervisor-level register accessible with the b.mtspr/b.mfspr instructions in supervisor mode. The DMR1 is programmed with the watchpoint/breakpoint settings that define how DVR/DCR pairs operate and is set by the resident debug software or by the development interface.

Bit	31 ... 25	23	22	21 20	19 18	17 16
Identifier	Reserved	Res	ST	Res	Res	Res
Reset	X	X	0	X	X	X
R/W	R	R	R/W	R	R	R

Bit	15 14	13 12	11 10	9 8	7 6	5 4	3 2	1 0
Identifier	CW7	CW6	CW5	CW4	CW3	CW2	CW1	CW0
Reset	0	0	0	0	0	0	0	0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

CW0	Chain Watchpoint 0
00	Watchpoint 0 = Match 0
01	Reserved
1x	Reserved
CW1	Chain Watchpoint 1
00	Watchpoint 1 = Match 1
01	Watchpoint 1 = Match 1 AND Match 0
1x	Reserved
CW2	Chain Watchpoint 2
00	Watchpoint 2 = Match 2
01	Reserved
1x	Reserved
CW3	Chain Watchpoint 3
00	Watchpoint 3 = Match 3
01	Watchpoint 3 = Match 3 AND Match2
1x	Reserved

CW4	Chain Watchpoint 4
00	Watchpoint 4 = Match 4
01	Reserved
1x	Reserved
CW5	Chain Watchpoint 5
00	Watchpoint 5 = Match 5
01	Watchpoint 5 = Match 5 AND Match4
1x	Reserved
CW6	Chain Watchpoint 6
00	Watchpoint 6 = Match 6
01	Reserved
1x	Reserved
CW7	Chain Watchpoint 7
00	Watchpoint 7 = Match 7
01	Watchpoint 7 = Match 7 AND Match6
1x	Reserved
ST	Single-step Trace
0	Single-step trace disabled
1	Every executed instruction causes trap exception

Table 26: DMR 1 Field Descriptions

10.5 Debug Mode Register 2 (DMR2)

The debug mode register 2 is a 32-bit special-purpose supervisor-level register accessible with the b.mtspr/b.mfspr instructions in supervisor mode. The DMR2 is programmed with the watchpoint/breakpoint settings that define which match conditions generate a breakpoint/watchpoint. When a breakpoint/watchpoint happens WBS provides information which or several match conditions caused breakpoint/watchpoint condition. WBS bits are sticky and should be cleared by writing 0 to them every time a breakpoint condition is processed. DMR2 is set by the resident debug software or by the development interface.

Bit	31 ... 22	21 ... 12	11 ... 2	1	0
Identifier	WBS	WGB	Res	Res	Res
Reset	0	0	X	X	X
R/W	R	R/W	R	R	R

`00 0000 1111`	First four watchpoints increment counter 1, rest increment counter 0
`XX 1111 1111`	All watchpoints increment counter 1
WGB	Watchpoints Generating Breakpoint (trap exception)
`XX 0000 0000`	Breakpoint disabled
`XX 0000 0001`	Watchpoint 0 generates breakpoint
`XX 1111 1111`	All watchpoints generate breakpoint
WBS	Watchpoints Breakpoint Status
`XX 0000 0000`	No watchpoint caused breakpoint
`XX 0000 0001`	Watchpoint 0 caused breakpoint
`XX 1111 1111`	Any watchpoint could have caused breakpoint

Table 27: DMR 2 Field Descriptions

10.6 Debug Stop Register (DSR)

The debug stop register is a 32-bit special-purpose supervisor-level register accessible with the b.mtspr/b.mfspr instructions in supervisor mode. The DSR specifies which exceptions cause the core to stop the execution of the exception handler and turn over control to development interface. It can be programmed by the resident debug software or by the development interface.

Bit	31 ... 14	13	12	11	10	9	8
Identifier	Reserved	TE	FPE	SCE	RE	IME	DME
Reset	X	0	0	0	0	0	0
R/W	R	R/W	R/W	R/W	R/W	R/W	R/W

Bit	7	6	5	4	3	2	1	0
Identifier	INTE	IIE	AE	TTE	IPFE	DPFE	BUSEE	RSTE
Reset	0	0	0	0	0	0	0	0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

RSTE	Reset Exception
0	This exception does not transfer control to the development I/F
1	This exception transfers control to the development interface
BUSEE	Bus Error Exception

0	This exception does not transfer control to the development I/F
1	This exception transfers control to the development interface
DPFE	Data Page Fault Exception
0	This exception does not transfer control to the development I/F
1	This exception transfers control to the development interface
IPFE	Instruction Page Fault Exception
0	This exception does not transfer control to the development I/F
1	This exception transfers control to the development interface
TTE	Tick Timer Exception
0	This exception does not transfer control to the development I/F
1	This exception transfers control to the development interface
AE	Alignment Exception
0	This exception does not transfer control to the development I/F
1	This exception transfers control to the development interface
IIE	Illegal Instruction Exception
0	This exception does not transfer control to the development I/F
1	This exception transfers control to the development interface
INTE	Interrupt Exception
0	This exception does not transfer control to the development I/F
1	This exception transfers control to the development interface
DME	DTLB Miss Exception
0	This exception does not transfer control to the development I/F
1	This exception transfers control to the

	development interface
IME	ITLB Miss Exception
0	This exception does not transfer control to the development I/F
1	This exception transfers control to the development interface
RE	Range Exception
0	This exception does not transfer control to the development I/F
1	This exception transfers control to the development interface
SCE	System Call Exception
0	This exception does not transfer control to the development I/F
1	This exception transfers control to the development interface
FPE	Floating Point Exception
0	This exception does not transfer control to the development I/F
1	This exception transfers control to the development interface
TE	Trap Exception
0	This exception does not transfer control to the development I/F
1	This exception transfers control to the development interface

Table 28: DSR Field Descriptions

10.7 Debug Reason Register (DRR)

The debug reason register is a 32-bit special-purpose supervisor-level register accessible with the b.mtspr/b.mfspr instructions in supervisor mode. The DRR specifies which event caused the core to stop the execution of program flow and turned control over to the development interface. It should be cleared by the resident debug software or by the development interface.

Bit	31 ... 14	13	12	11	10	9	8
Identifier	Reserved	TE	FPE	SCE	RE	IME	DME
Reset	X	0	0	0	0	0	0
R/W	R	R/W	R/W	R/W	R/W	R/W	R/W

Bit	7	6	5	4	3	2	1	0
Identifier	INTE	IIE	AE	TTE	IPFE	DPFE	BUSEE	RSTE
Reset	0	0	0	0	0	0	0	0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

RSTE	Reset Exception
0	This exception did not transfer control to the development I/F
1	This exception transfered control to the development interface
BUSEE	Bus Error Exception
0	This exception did not transfer control to the development I/F
1	This exception transfered control to the development interface
DPFE	Data Page Fault Exception
0	This exception did not transfer control to the development I/F
1	This exception transfered control to the development interface
IPFE	Instruction Page Fault Exception
0	This exception did not transfer control to the development I/F
1	This exception transfered control to the development interface
TTE	Tick Timer Exception
0	This exception did not transfer control to the development I/F
1	This exception transfered control to the development interface
AE	Alignment Exception

0	This exception did not transfer control to the development I/F
1	This exception transfered control to the development interface
IIE	Illegal Instruction Exception
0	This exception did not transfer control to the development I/F
1	This exception transfered control to the development interface
INTE	Interrupt Exception
0	This exception did not transfer control to the development I/F
1	This exception transfered control to the development interface
DME	DTLB Miss Exception
0	This exception did not transfer control to the development I/F
1	This exception transfered control to the development interface
IME	ITLB Miss Exception
0	This exception did not transfer control to the development I/F
1	This exception transfered control to the development interface
RE	Range Exception
0	This exception did not transfer control to the development I/F
1	This exception transfered control to the development interface
SCE	System Call Exception
0	This exception did not transfer control to the development I/F
1	This exception transfered control to the development interface
FPE	Floating Point Exception
0	This exception did not transfer control to the development I/F
1	This exception transferred control to the

	development interface
TE	Trap Exception
0	This exception did not transfer control to the development I/F
1	This exception transferred control to the development interface

Table 29: DRR Field Descriptions

11 Programmable Interrupt Controller (Optional)

This chapter describes the BA2 level one programmable interrupt controller. The interrupt controller facility is optional and the developer may choose whether or not to include it. UPR[PICP] specifies whether the programmable interrupt controller is implemented or not. If it is not implemented, interrupt input is directly connected to interrupt exception inputs. The Programmable Interrupt Controller has two special-purpose registers and 32 maskable interrupt inputs.

11.1 Features

The BA2 architecture defines an interrupt controller facility with up to 32 interrupt inputs:

11.2 PIC Mask Register (PICMR)

The interrupt controller mask register is a 32-bit special-purpose supervisor-level register accessible with the b.mtspr/b.mfspr instructions in supervisor mode. PICMR is used to mask or unmask 32 programmable interrupt sources.

Bit	31-0
Identifier	IUM
Reset	0
R/W	R/W

IUM	Interrupt UnMask
`0x00000000`	All interrupts are masked
`0x00000001`	Interrupt input 0 is enabled, all others are masked
`0xFFFFFFFF`	All interrupt inputs are enabled

Table 30: PICMR Field Descriptions

11.3 PIC Status Register (PICSR)

The interrupt controller status register is a 32-bit special-purpose supervisor-level register accessible with the b.mtspr/b.mfspr instructions in supervisor mode. PICSR is used to determine the status of each PIC interrupt input. PIC only supports level-triggered interrupts where bits in PICSR simply represent level of interrupt inputs. Interrupts are cleared by taking appropriate action at the device to negate the source of the interrupt. Writing a '1' or a '0' to bits in the PICSR that reflect a level-triggered source must have no effect on PICSR content.

Bit	31-0
Identifier	IS
Reset	0
R/W	R/(W*)

IS	Interrupt Status
`0x00000000`	All interrupts are inactive
`0x00000001`	Interrupt input 0 is pending
`0xFFFFFFFF`	All interrupts are pending

Table 31: PICSR Field Descriptions

12 Tick Timer Facility (Optional)

This chapter describes the BA2 tick timer facility. It is optional and an implementation may chose whether or not to implement it. UPR[TTP] specifies whether or not the tick timer facility is present. The Tick Timer is used to schedule operating system and user tasks on regular time basis or as a high precision time reference. The Tick Timer facility is enabled with TTMR[M]. TTCR is incremented with each clock cycle and a tick timer interrupt can be asserted whenever the lower 28 bits of TTCR match TTMR[TP] and TTMR[IE] is set. TTCR restarts counting from zero when a match event happens and TTMR[M] is 0x1. If TTMR[M] is 0x2, TTCR is stopped when match event happens and TTCR must be changed to start counting again. When TTMR[M] is 0x3, TTCR keeps counting even when match event happens.

12.1 Features

The BA2 architecture defines a tick timer facility with the following features:

- Maximum timer count of 2^32 clock cycles
- Maximum time period of 2^28 clock cycles between interrupts
- Maskable tick timer interrupt
- Single run, restartable counter, or continues counter

12.2 Timer Interrupts

A timer interrupt will happen every time TTMR[IE] bit is set and TTMR[TP] matches the lower 28-bits of the TTCR SPR - the top 4 bits are ignored for the comparison. When an interrupt is pending the TTMR[IP] bit will be set and the interrupt will be asserted to the CPU core until it is cleared by writing a 0 to the TTMR[IP] bit. However, if the TTMR[IE] bit was not set when a match condition occurred no interrupt will be asserted and the TTMR[IP] bit won't be set unless it has not been cleared from a previous interrupt. The TTMR[IE] bit is not meant as a mask bit, SR[TEE] is provided for that purpose.

12.3 Timer Modes

It is up to the programmer to ensure that the TTCR SPR is set to a sane value before the timer mode is programmed. When the timing mode is programmed into the timer by setting TTMR[M], the TTCR SPR is not preset to any predefined value, including 0. If the lower 28-bits of the TTCR SPR is numerically greater than what was programmed into TTMR[TP] then the timer will only assert the timer interrupt when the lower 28-bits of the TTCR SPR have wrapped around to 0 and counted up to the match value programmed into TTMR[TP].

12.3.1 Disabled Timer

In this mode the timer does not increment the TTCR SPR. Though note that the timer interrupt is independent from the timer mode and as such the timer interrupt is not disabled when the timer is disabled.

12.3.2 Auto-restart Timer

When the timer is set to auto-restart mode, the timer will reset the TTCR SPR to 0 as soon as the lower 28-bits of the TTCR SPR match TTMR[TP] and the timer interrupt will be asserted to the CPU core if the TTMR[IE] bit has been set.

12.3.3 One-shot Timer

In one-shot timing mode, the timer stops counting as soon as a match condition has been reached. Although the timer has in effect been disabled (and can't be restarted by writing to the TTCR SPR) the TTMR[M] bits shall still indicate that the timer is in one-shot mode and not that it has been disabled. Care should be taken that the timer interrupt has been masked (or disabled) after the match condition has been reached, or else the CPU core will get a spurious timer interrupt.

12.3.4 Continuous Timer

In the event that a match condition has been reached, the counter does not stop but rather keeps counting from the value of the TTCR SPR and the timer interrupt will be asserted if the TTMR[IE] bit has been set.

12.4 Tick Timer Mode Register (TTMR)

The tick timer mode register is a 32-bit special-purpose supervisor-level register accessible with the b.mtspr/b.mfspr instructions in supervisor mode. The TTMR is programmed with the time period of the tick timer as well as with the mode bits that control operation of the tick timer.

Bit	31 30	29	28	27 ... 0
Identifier	M	IE	IP	TP
Reset	0	0	0	X
R/W	R/W	R/W	R	R/W

TP	Time Period
`0x0000000`	Shortest comparison time period
`0xFFFFFFF`	Longest comparison time period
IP	Interrupt Pending
0	Tick timer interrupt is not pending
1	Tick timer interrupt pending (write '0' to clear it)
IE	Interrupt Enable
0	Tick timer does not generate tick timer interrupt
1	Tick timer generates tick timer interrupt when TTMR[TP] matches TTCR[27:0]
M	Mode
00	Tick timer is disabled
01	Restart - Timer is restarted when TTMR[TP] matches TTCR[27:0]
10	Single Run - Timer stops when TTMR[TP] matches TTCR[27:0] (change TTCR to resume counting)

11	Continuous Run - Timer does not stop when TTMR[TP] matches TTCR[27:0]

Table 32: TTMR Field Descriptions

12.5 Tick Timer Count Register (TTCR)

The tick timer count register is a 32-bit special-purpose register accessible with the b.mtspr/b.mfspr instructions in supervisor mode. TTCR holds the current value of the timer.

Bit	31-0
Identifier	CNT
Reset	0
R/W	R/W

CNT	Count
	32-bit incrementing counter

Table 33: TTCR Field Descriptions

12.6 Tick Timer ReStart Register (TTRS)

The Tick timer ReStart register is a 32-bit special purpose register accessible with the b.mtspr/b.mfspr instructions in supervisor mode. When the register is read it clears the TTMR[IP] and when it is written it sets a new period (the value being written) to TTMR[TP] and resets the TTCR to 0 when the mode TTMR[M] is set to “single-run mode”. When the mode is not set to “single-run” the result of a write to TTRS is undefined and should be avoided.

Bit	31-0
Identifier	PERIOD
Reset	0
R/W	R/W

Table 34: TTRS Field Descriptions

13 Power Management Facility (Optional)

This chapter describes the BA2 power management facility. Implementation is optional and can be selected using compile time parameters file. UPR[PM] specifies whether or not the power management facility is present.

13.1 Features

13.1.1 Power Management I/O Interface

System uses the interface to request facility's attention and monitor its response. Facility uses the interface to respond to system's requests and/or signal internal events to the system. System uses the interface's stall input signal to request attention. The facility then puts CPU into idle state in which external clock manipulation (gating, speed change) is allowed. This state is signaled on the interface's stalled output signal to allow the system to proceed with intended operation. The facility will keep the CPU in idle state until the system releases stall signal. A special power management event signal can be software configured to propagate Tick Timer's and/or Programmable Interrupt Controller's active interrupt requests to the system. This can be used for wake-up operations or simply to get attention from the system when idle.

13.1.2 Configurable Power Management Event

Software configures power management event signal usage via Power Management Event Control register (PMEVNT).

13.1.3 Hardware Controlled Register File Clock Gating

Separate clock input and clock enable output signals are provided for register file registers. This enables clock gating for every clock cycle during which no writes occur to register file.

13.1.4 Software Controlled Clock Gating

Some processor units do not require running clock signal when not in use. These units have its own clock input along with clock enable signal provided. Clock enable signals are under software control using Power Management Unit(s) Power Down register (PMUNITPD).

13.2 Power Management Event Control register (PMEVNT)

The power management event control register is a 32-bit special-purpose supervisor-level register accessible with the b.mtspr/b.mfspr instructions in supervisor mode. Used to configure power management event output signal behavior while the processor is stalled via power management IO interface.

Bit	1	0
Identifier	PMETTEN	PMEPICEN
Reset	0	0
R/W	R/W	R/W

PMETTEN	Tick Timer Power Management Event Enable
0	Tick timer interrupt does not trigger power management events
1	Tick timer interrupt triggers power management events
PMEPICEN	Programmable Interrupt Controller Power Management Event Enable
0	Programmable interrupt controller interrupt does not trigger power management events
1	Programmable interrupt controller interrupt triggers power management events

Table 35: PMEVNT Field Descriptions

13.3 Power Management Unit(s) Power Down register (PMUNITPD)

The power management unit(s) power down register is a 32-bit special-purpose supervisor-level register accessible with the b.mtspr/b.mfspr instructions in supervisor mode. Used to control clock enable signals for individual units.

Bit	0
Identifier	PMDUCLKEN
Reset	1
R/W	R/W

PMDUCLKEN	Debug Unit Clock Enable
0	Debug unit clock enable output signal de-asserted
1	Debug unit clock enable output signal asserted

Table 36: PMUNITPD Field Descriptions

14 Memory management (Optional)

This chapter describes the virtual memory and access protection mechanisms for memory management within the BA2 architecture.

Note that this chapter describes the address translation mechanism from the perspective of the programming model. As such, it describes the structure of the page tables, the MMU conditions that cause MMU related exceptions and the MMU registers. The hardware details that are invisible to the BA2 programming model, such as MMU organization and TLB size, are not contained in the architectural definition.

14.1 Overview

The primary functions of the MMU in BA2 processors are to translate effective addresses to physical addresses for memory accesses. In addition, the MMU provides various levels of access protection on a page-by-page basis. Note that this chapter describes the conceptual model of the BA2 MMU and processors may differ in the specific hardware used to implement this model.

Two general types of accesses generated by BA2 processors require address translation – instruction accesses generated by the instruction fetch unit, and data accesses generated by the load and store unit. Generally, the address translation mechanism is defined in terms of page tables used by BA2 processors to locate the effective to physical address mapping for instruction and data accesses.

The definition of page table data structures provides significant flexibility for the processor of performance enhancement features in a wide range of processors. Therefore, the performance enhancements used to the page table information on-chip vary from processor to processor.

Translation lookaside buffers (TLBs) are commonly implemented in BA2 processors to keep recently-used page address translations on-chip. Although their exact implementation is not specified, the general concepts that are pertinent to the system software are described.

The MMU, together with the exception processing mechanism, provides the necessary support for the operating system to implement a paged virtual memory environment and for enforcing protection of designated memory areas.

14.2 Memory management exceptions

To complete any memory access, the effective address must be translated to a physical address. An MMU exception occurs if this translation fails.

TLB miss exceptions happen on BA2 processors when a TLB miss occurs and software intervention is required to replace TLB entry with an entry from the page table.

The page fault exceptions that are caused by missing PTE in page table or page access protection can happen on any BA2 processor.

Exception Type	Vector Offset	Causal Conditions
Data Page Fault	0x300	No matching PTE found in page tables or page protection violation for load/store operations.
Instruction Page Fault	0x400	No matching PTE found in page tables or page protection violation for instruction fetch.
D-TLB Miss	0x900	No matching entry in DTLB (DTLB miss).
I-TLB Miss	0xA00	No matching entry in ITLB (ITLB miss).

Table 37: MMU Exception

The state saved by the processor for each of the exceptions in Table 37 contains information that identifies the address of the failing instruction. Refer to the chapter Exception Model for a more detailed description of exception processing.

14.3 MMU Special-Purpose Registers

Table 38 summarizes the registers that the operating system uses to program the MMU. These registers are 32-bit special-purpose supervisor-level registers accessible with the b.mtspr/b.mfspr instructions in supervisor mode. Access to MMU registers is allowed only when corresponding MMU is disabled by having its SR[DME] or SR[IME] bit cleared. Access to MMU registers while corresponding MMU is enabled, is undefined.

Table 38 does not show two configuration registers that are implemented if processor implements configuration registers. DMMUCFGR and IMMUCFGR describe capability of DMMU and IMMU.

Grp #	Reg #	Reg Name	USER MODE	SUPV MODE	Description
1	2	DTLBEIR	–	W	Data TLB Entry Invalidate register
1	8	DTLBMR2	-	W	Data TLB Match register
1	10	DTLBTR2	-	W	Data TLB Translate register
2	2	ITLBEIR	–	W	Instruction TLB Entry Invalidate register
2	8	ITLBMR2	-	W	Instruction TLB Match register
2	10	ITLBTR2	-	W	Instruction TLB Translate register

Table 38: MMU Special purpose registers

As TLBs are noncoherent caches of PTEs, software that changes the page tables in any way must perform the appropriate TLB invalidate operations to keep the on-chip TLBs coherent with respect to the page tables in memory.

14.4 Instruction / Data TLB Entry Invalidate Registers (xTLBEIR)

The instruction/data TLB entry invalidate registers are special-purpose registers accessible with the b.mtspr/b.mfspr instructions in supervisor mode.

The xTLBEIR is written with the effective address. The corresponding xTLB entry is invalidated in the local processor.

Bit	31-1	0
Identifier	EA	IH
Reset	0	0
R/W	Write Only	W

IH	Invalidate if Hit
0	Invalidate TLB entry
1	Invalidate TLB entry only if EA hits in the TLB
EA	Effective Address

Table 39: xTLBEIR Field Descriptions

14.5 Instruction / Data Translation Lookaside Buffer March Register (xTLBMR2)

The xTLBMR register is 32-bit special-purpose supervisor-level register accessible with the b.mtspr/b.mfspr instructions in supervisor mode.

xTLBMR register is an interface to xTLB match RAM that hold tags. A write to xTLBMR will automatically select and write to the least-recently used way and xTLB set.

Bit	31-13
Identifier	VPN
Reset	X
R/W	W

VPN	Virtual Page Number 0-N Number of the virtual frame that must match EA

Table 40: xTLBMR2 Field Descriptions

14.6 Data Translation Lookaside Buffer Translate Register (DTLBTR2)

DTLBTR register is an interface to portion of DTLB that holds translated physical address and attributes. A write to DTLBTR has to be preceded by a write to DTLBMR, which will select the correct DTLB set and way.

Bit	31-13	12-5	4	3	2	1	0
Identifier	PPN	Reserved	W	R	U	CI	P
Reset	X	X	X	X	X	X	X
R/W	W	W	W	W	W	W	W

P	Present
0	Data TLB disabled (writes get ignored)
1	Data TLB enabled
CI	Cache Inhibit
0	Cache is enabled for this page
1	Cache is disabled for this page
U	Everything that the supervisor can do on the page, userspace can also do
0	Disabled
1	Enabled
R	Supervisor mode can read the page
0	Disabled
1	Enabled
W	Supervisor mode can write the page
0	Disabled
1	Enabled
PPN	Phyisical Page Number 0-N Number of the Physical frame to which VPN is translated

Table 41: DTLBTR Field Descriptions

14.7 Instruction Translation Lookaside Buffer Translate Register (ITLBTR2)

The ITLBTR registers is 32-bit special-purpose supervisor-level register accessible with the b.mtspr/b.mfspr instructions in supervisor mode.

ITLBTR register is an interface to portion of ITLB that holds translated physical address and attributes. A write to ITLBTR has to be preceded by a write to ITLBMR, which will select the correct ITLB set and way.

Bit	31-13	12-6	5	4-3	2	1	0
Identifier	PPN	Reserved	E	Reserved	U	CI	P
Reset	X	X	X	X	X	X	X
R/W	W	W	W	W	W	W	W

P	Present
0	Instruction TLB disabled (writes get ignored)
1	Instruction TLB enabled
CI	Cache Inhibit
0	Cache is enabled for this page
1	Cache is disabled for this page
U	Everything that the supervisor can do on the page, userspace can also do
0	Disabled
1	Enabled
E	Supervisor mode can execute the page
0	Disabled
1	Enabled
PPN	Phyisical Page Number 0-N Number of the Physical frame to which VPN is translated

Table 42: ITLBTR Field Descriptions

14.8 Virtual Memory and Page Protection Concept

Since BA2 processors implement software TLB reload, organization of page tables and page table entries (PTE) is not defined. It is up to the operating system to use whatever organization of page tables and PTEs that is most optimal for a given operating system or firmware. However basic concept of address translation and page protection is described to provide a general overview.

14.9 Memory Protection Mechanism

After a virtual address is determined to be within a page covered by the valid PTE, the access is validated by the memory protection mechanism. If this protection mechanism prohibits the access, a page fault exception is generated.

The memory protection mechanism allows selectively granting read access, write access or execute access for both supervisor and user modes. The page protection mechanism provides protection at all page level granularities.

Protection attribute	Meaning
DTLBTR[SREx]	Enable load operations in supervisor mode to the page.
DTLBTR[SWEx]	Enable store operations in supervisor mode to the page.
ITLBTR[SXEx]	Enable execution in supervisor mode of the page.
DTLBTR[UREx]	Enable load operations in user mode to the page.
DTLBTR[UWEx]	Enable store operations in user mode to the page.
ITLBTR[UXEx]	Enable execution in user mode of the page.

Table 43: Protection Attributes

14.10 Page Table Operations

Two TLB miss exceptions are used to handle TLB reload operations. TLBs are noncoherent caches of the page tables and must be maintained accordingly. Explicit software syncronization between TLB and page tables is required so that page tables and TLBs remain coherent.

Updates to the page tables include operations like adding a PTE, deleting a PTE and modifying a PTE. On multiprocessor systems exclusive access to the page table must be assured before it is modified.

Since the processor reloads PTEs even during updates of the page table, special care must be taken when updating page tables so that the processor does not accidently use half modified page table entries.

15 Cache Model (Optional)

This chapter describes the BA2 cache model and architectural control to maintain cache coherency in multiprocessor environment.

Note that this chapter describes the cache model and cache coherency mechanism from the perspective of the programming model. As such, it describes the cache management principles, the cache coherency mechanisms and the cache control registers. The hardware processor details that are invisible to the BA2 programming model, such as cache organization and size, are not contained in the architectural definition.

The function of the cache management registers depends on the processor of the cache(s) and the setting of the memory/cache access attributes. For a program to execute properly on all BA2 processor processors, software should assume a Harvard cache model. In cases where a processor is implemented without a cache, the architecture guarantees that writing to cache registers will not halt execution. For example a processor without cache should simply ignore writes to cache management registers. In this manner, programs written for separate instruction and data caches will run on all compliant processors.

15.1 Cache Special-Purpose Registers

Table 44 summarizes the registers that the operating system uses to manage the cache(s). Access to cache registers is allowed only when corresponding cache is disabled by having its SR[DCE] or SR[ICE] cleared. Access to cache registers while corresponding cache is enabled, is undefined.

For processors that have unified cache, registers that control the data and instruction caches are merged and available at the same time both as data and instruction cache registers.

GRP #	REG #	REG NAME	USER MODE	SUPV MODE	DESCRIPTION
3	2	DCBFR	–	W	Data Cache Block Flush Register
3	3	DCBIR	–	W	Data Cache Block Invalidate Register
4	2	ICBIR	–	W	Instruction Cache Block Invalidate Register

Table 44: Cache Registers

15.2 Data Cache Block Invalidate (DCBIR)

The data cache block invalidate register is a special-purpose register accessible with the b.mtspr/b.mfspr instructions in supervisor mode.

Bit	31-1	0
Identifier	EA	IH
Reset	0	0
R/W	Write Only	Write Only

IH	Invalidate if Hit
0	Invalidate cache block
1	Invalidate cache block only if EA hits in the cache
EA	Effective Address

Table 45: DCBIR Field Descriptions

15.3 Data Cache Block Flush (DCBFR)

The data cache block flush register is a special-purpose register accessible with the b.mtspr/b.mfspr instructions in supervisor mode. Flushing registers in write through caches as implemented in BA2 architectures is equal to invalidation.

Bit	31-1	0
Identifier	EA	IH
Reset	0	0
R/W	Write Only	Write Only

IH	Invalidate and writeback if Hit
0	Invalidate and writeback cache block
1	Invalidate and writeback cache block only if EA hits in the cache
EA	Effective Address

Table 46: DCBIR Field Descriptions

15.4 Instruction Cache Block Invalidate

The instruction cache block invalidate register is a special-purpose register accessible with the b.mtspr/b.mfspr instructions in supervisor modes.

The ICBIR is written with the effective address causing the instruction cache block is invalidated.

Bit	31-1	0
Identifier	EA	IH
Reset	0	0
R/W	Write Only	Write Only

IH	Invalidate if Hit
0	Invalidate cache block
1	Invalidate cache block only if EA hits in the cache
EA	Effective Address

Table 47: ICBIR Field Descriptions

15.5 Cache memory coherency

The primary role of the cache coherency system is to synchronize cache content with other caches and with the memory and to provide the same image of the memory to all devices using the memory.

The architecture provides several features to implement cache coherency. In systems that do not provide cache coherency with the PTE attributes (because they do not implement a memory management unit), it may be provided through explicit cache management.

Cache coherency in systems with virtual memory can be provided on a page-by-page basis with PTE attribute Caching-Inhibited (CI Attribute).

When the memory/cache attributes are changed, it is imperative that the cache contents should reflect the new attribute settings. This usually means that cache blocks must be flushed or invalidated.

15.6 Pages designated as Caching-Inhibited Pages

Memory accesses to memory pages designated with CI=1 are always performed directly into the main memory, bypassing all caches. Memory pages designated with CI=1 are not loaded into the cache and the target content should never be available in the cache. To prevent any accident copy of the target location in the cache, whenever the operating system sets a memory page to be caching-inhibited, it should flush the corresponding cache blocks.