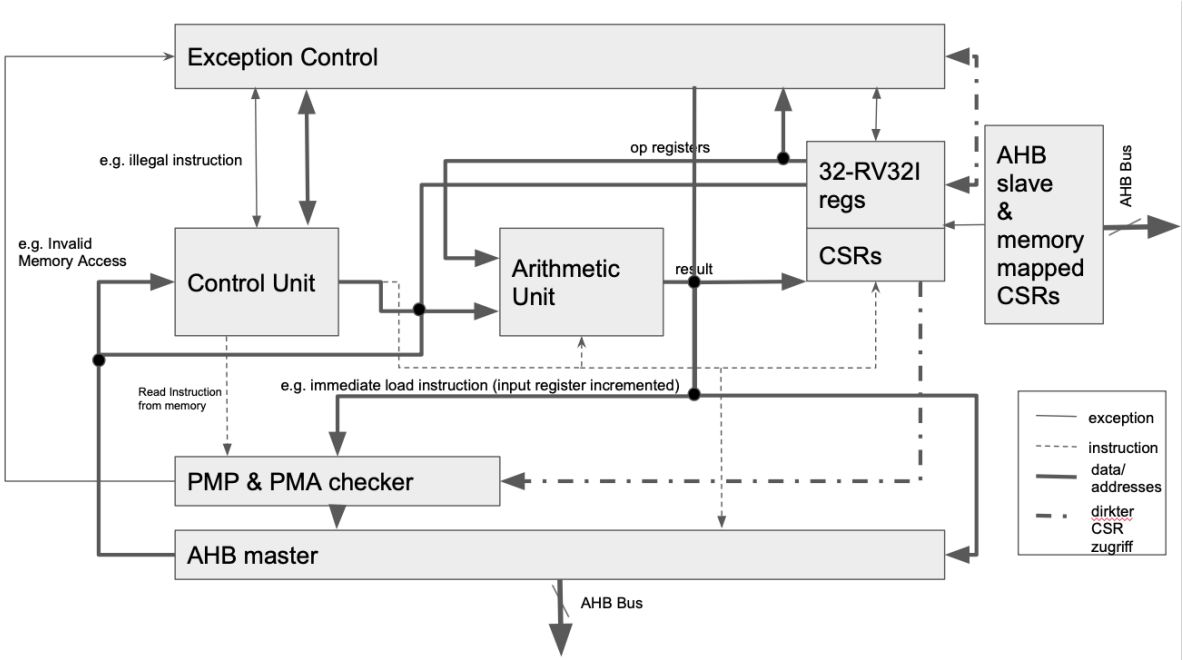
System Design:

The scope of this document is to describe the data path and components of the system shown below:



Data Path

There are three different types of signals, data/address signals, control signals and exception-control signals.
Control signals and exception-control signals will be covered in the chapters “Exception Control” and “Control Unit”
In order to process data it needs to be stored inside the Register Unit, the processor does not support atomic operations like read-modify-write.

Instruction Fetches:

- The PC is loaded through the ALU to the PMP & PMA checker
- If the addresses passes the PMP & PMA checker, the load is performed
- After the AHB master receives the instruction, it is stored in the Instruction Register of the Control Unit.

The basic Data Path for arithmetic operations is the following:

- Values are read from the Register Unit
- If one value is an immediate it is decoded by the control unit and forwarded to one of the inputs of the Arithmetic Unit
- The Arithmetic Unit performs some operation
- Data is written to the Register Unit

If a load is performed:

- The address value is computed by a combination of the base value (register) and an immediate, this is done using the alu
- instead of writing the value back to the register, it is loaded into the PMP & PMA
- if the address passes the PMP & PMA checker, the memory access is performed
- After the AHB master receives the value, it is loaded into the Register Unit using the alu to perform zero & signed extension (optional on LH, LHU, LB & LBU)

If stores are performed,:

- the address is generated and checked (load)
- During address check the store value is read from the Register Unit masked inside the alu (optional for SB, SH) and buffered inside the AHB unit.
- if the address passes the PMP & PMA check, the store is performed

Exception Control

The Exception Control unit monitors Exception Signals and the mip bit for pending interrupts. Exceptions can be thrown by the PMP & PMA checker and the Control Unit..
If an Exception or Interrupt is raised, the Exception Control unit must write the current PC+4 to the mpec register and modify the mcause register to reflect the cause of the exception/interrupt.
Also, mtval must be written as specified in “Version1_privileged_requirements.pdf” on page 11 & 12.
If multiple exceptions occur at once, only the highest priority exception is taken.
The priority in which the exceptions are taken is specified in “Version1_privileged_requirements.pdf” on page 11.
If an Exception is raised, while the processor is handling another exception, mpec, mcause & mtval is overwritten. The saving of the return address and other information stored in those registers is left to the trap handler, in order to avoid a large hardware register stack.

The following Interrupts may occur:

- Timer Interrupts
- Software Interrupts

Timer interrupts are set to pending, if the mtimecmp register equals the mtime register.
The Machine Mode Software Interrupt Pending bit can be set by software to cause a software interrupt.

The following exceptions may occur:

- illegal instruction exception
- breakpoint exception
- environment-call-from-M-mode exception
- load access fault exception
- store/AMO access fault exception
- instruction access fault exception
- load address misaligned exception
- store/AMO address misaligned exception
- instruction address misaligned exception

illegal instruction exception:

- if access is performed to a non implemented CSR
- write a read-only csr
- if software tries to write an illegal value to WLRL CSR
- illegal/unknown instructions

breakpoint exception:

- caused by EBREAK

environment-call-from-M-mode exception

- caused by ECALL if executed in M-mode

load/(store/AMO)/instruction address/access misaligned/fault

- caused by PMP & PMA checker

The exception control unit needs access to the following registers:

- PC (read, write)
- mtval (write)
- mtvec (read)
- mcause (write)
- mie & mip (inside the CSR unit, mie & mip bits are logical linked, and the signal is given as an exception signal to the EC)
- mepc (write)

Exception Occurs:

- check if interrupt is enabled
- stop core
- load current pc+4 to mepc
- modify mcause register to reflect the exception
- load mtval with additional information
- set PC according to mtvec
- start core

Control Unit

The control Unit is the part of the CPU that decodes given instructions and instructs the corresponding CPU parts. The control unit has to be able to decode every possible instruction, load fetch information from the memory and send instructions to the different CPU units.

Inputs:

- Program Counter write/read
- Instruction Register
- enable bit
- reset

Exceptions: Listening for signals coming from the exception control unit. On exception call immediately stop the core and wait for further signals of exception control.

Illegal Instruction: The control unit checks whether the given instruction is valid for this CPU. If the instruction is illegal the exception control is activated.

Instructions: The Instructions to be implemented are listed in the following table:

Register-Register Instructions:

mnemonic	destination register	source 1 register	source 2 register	description
ADD	xD	xA	xB	Add
SUB	xD	xA	xB	Subtract
AND	xD	xA	xB	AND
OR	xD	xA	xB	OR
XOR	xD	xA	xB	XOR
SLT	xD	xA	xB	Set Less Than
SLTU	xD	xA	xB	Set Less Than Unsigned
SLL	xD	xA	xB	Shift Left Logical
SRL	xD	xA	xB	Shift Right Logical
SRA	xD	xA	xB	Shift Right Arithmetic

Immediate Instructions

mnemonic	destination register	source 1 register	immediate	description
ADDI	xD	xA	CONSTANT	Add Immediate
ANDI	xD	xA	CONSTANT	AND Immediate
ORI	xD	xA	CONSTANT	OR Immediate
XORI	xD	xA	CONSTANT	XOR Immediate
SLTI	xD	xA	CONSTANT	Set Less Than Immediate
SLLI	xD	xA	SHIFT	Shift Left Logical Immediate
SRLI	xD	xA	SHIFT	Shift Right Logical Immediate
SRAI	xD	xA	SHIFT	Shift Right Arithmetic Immediate

Conditional branches

mnemonic	source 1 register	source 2 register	immediate	description
BEQ	xA	xB	OFFSET	Branch xA = xB
BNE	xA	xB	OFFSET	Branch xA != xB
BLT	xA	xB	OFFSET	Branch xA < xB
BGE	xA	xB	OFFSET	Branch xA >= xB
BLTU	xA	xB	OFFSET	Branch xA < xB Unsigned
BGEU	xA	xB	OFFSET	Branch xA >= xB Unsigned

Upper immediate instructions

mnemonic	destination register	immediate		description
LUI	xD	UPPER		Load Upper Immediate
AUIPC	xD	UPPER		Add Upper Immediate to PC

Unconditional jumps

mnemonic	destination register	immediate		description
JAL	xD	OFFSET		Jump AND Link

Indirect jump instruction

mnemonic	destination register	source 1 register	immediate	description
JALR	xD	xA	OFFSET	Jump and Link Register

Load and store instruction

mnemonic	destination register	immediate	source 1 register	description
LB	xD	OFFSET	(xB)	Load Byte
LH	xD	OFFSET	(xB)	Load Half
LW	xD	OFFSET	(xB)	Load Word
LBU	xD	OFFSET	(xB)	Load Byte Unsigned
LHU	xD	OFFSET	(xB)	Load Half Unsigned

mnemonic	source 2 register	immediate	source 1 register	description
SB	xS	OFFSET	(xB)	Store Byte
SH	xS	OFFSET	(xB)	Store Half
SW	xS	OFFSET	(xB)	Store Word
SBU	xS	OFFSET	(xB)	Store Byte Unsigned
SHU	xS	OFFSET	(xB)	Store Half Unsigned

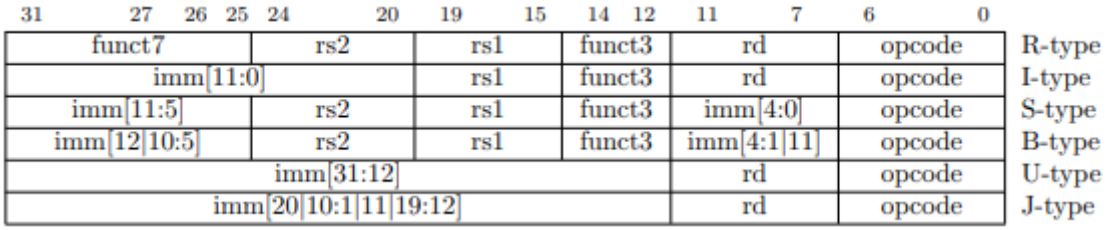
FENCE	IORW	IORW	Fence
ECALL			Service Request to execution environment
EBREAK			return control to debugging environment

CSRRW, CSRRS, CSRRC, CSRRWI, CSRRSI, CSRRCI
Information for CSR Instructions can be found in ISA Chapter 9 page 73.

There are different types of instructions as visible in the table above. The different types that have to be implemented are:

Instruction Type	Overview	Example
R-Type	OpCode xD, xA, xB	ADD
I-Type	OpCode xD, xA	LW, ADDI
S-Type	OpCode xA, xB	SW
B-Type	OpCode xA	BREQ
U-Type	OpCode xA, Immediate	LUI
J-Type	OpCode Immediate	JAL

Decoding: To Decode the corresponding OpCode, the underlying instruction structure has to be known. Since the different types of instructions are fixed, the structure can be implemented as shown in the following table:



The decoder itself requires only 3 inputs which are:

- clock
- enable bit
- instruction stream

Controlling: After Decoding the instructions, the control unit tells the different CPU units what to do. The control unit communicates with the ALU and the Register Unit as well as the Memory.
Therefore the outputs of the decoder are:

- ALU operation
- immediate value from instruction
- write enable bit (does the instruction write to the memory?)
- 3 register select signals

Arithmetic Unit

The arithmetic unit performs mathematical and logical operations based on the instruction it gets from the control unit.
The ALU requires following inputs:

- ALU operation (from the Control Unit)
- clock
- enable bit
- input A
- input B

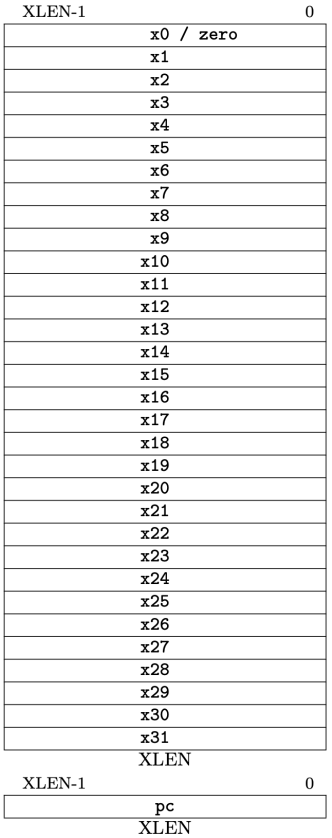
Based on the instruction, the ALU performs the operation and outputs following values:

- Operation Result
- Branch Flag (tells the control unit to skip the next instruction)

Register Unit

RV32I Registers

RV32I defines the following registers:



There are 31 general purpose registers (x1-x31), excluding register x0 which is hardwired to zero.
The PC is an additional register and placed inside the Control unit for this implementation

CSRs

CSRs are Control and Status Registers that are introduced to the design by the RISC-V privileged spec. Some of them need to be provided to the Exception Unit and the PMP & PMA checker via direct access. Therefore each register providing direct access will use 32 wires.
Some of the CSRs are memory mapped. Memory mapped means they need to be addressable via the system bus by any other AHB master. In order to implement this, a AHB salve module implementing all memory mapped CSRs is added to the design.
Further Information about this module can be found in the chapter “AHB Slave & memory-mapped CSRs”

The CSRs that provide direct access are:

- mpi (MTIP set by the “AHB slave & memory mapped CSRs” & to exception control (read))
- mie (read & write)
- mstatus (read)
- pmpcfg (read)
- pmpaddress (read)

non-memory-mapped CSRs (MXLEN = 32-bit):

register	addresse	type	access	width	description
misa	0x301	WARL	R	MXLEN	describes supported ISAs
mvendorid	0xF11	-	R	32-bit	describes vendor id
marchid	0xF12	-	R	MXLEN	describes architecture ID
mimpid	0xF13	-	R	MXLEN	describes implementation ID
mhartid	0xF14	-	R	MXLEN	describes hart id
mstatus	0x300	-	R/W	MXLEN	reflects & controls a hart’s current operating state
mtvec	0x305	-	R/W	MXLEN	holds trap vector configuration
mie	0x304	-	R/W	MXLEN	reflects interrupt enable state
mip	0x344	-	R	MXLEN	holds interrupt pending bits
mcycle	0xB00	-	R/W	64-bit	holds count of clock cycles
minstret	0xB02	-	R/W	64-bit	holds count of executed instructions
mhpmcounter (3-31)	0xB03-0xB1F	-	R/W	32-bit	holds count of events (lower 32 bit)
mhpcounterh(3-31)	0xB83-0xB9F		R/W		holds counter of events (upper 32 bit)
mhpmevent (3-31)	0x323-0x33F	-	R/W	MXLEN	specifies events on which to increment corresponding mhpmcounter

mcountinhibit	0x320	WARL	R/W	32-bit	controls which hardware performance monitoring counters increment (if set, no incremnt)
mscratch	0x340	-	R/W	MXLEN	Dedicated for use by machine-mode
mepc	0x341	WARL	R/W	MXLEN	holds jump-back address during interrupt
mcause	0x342	-	R/W	MXLEN	specifies cause of exception/interrupt
mtval		-	R/W	MXLEN	holds information about trap
pmpcfg0-3	0x3A0-0x3A3	-	R/W	MXLEN	Physical memory protection configuration
pmpaddr0-15	0x3B0-0x3BF	-	R/W	MXLEN	Physical memory protection address register

The Performance Monitor counter CSRs each need a seperate incrementing unit, in order to generate information about the CPU performance, without decreasing it.

PMP & PMA checker

The PMP and PMA checker is a unit that protects the memory. the PMA defines a set of attributes for each memory region

PMP

The PMP (Physical Memory Protection) Unit is used to define memory regions. It is only applied, if the PMP registers are locked. They can only be unlocked by a system reset.
If a PMP entry is not locked, every memory access that matches this address space may succeed.
The PMP needs information from the Control Unit, if the access is a read, write or fetch.
More information about the PMP and the required CSRs can be found in “Version1_privileged_requirements”.
Each PMP check shall be performed in one clock cycle. To allow this, the PMP needs direct access to its dedicated CSRs.

PMA checks

The only PMP check that must be implemented is to check if a memory address is misaligned.

Atomicity PMAs

The Implementation will not support the “A” extension, therefore this check will always be true.

Memory Ordering PMAs

Since there is no pipeline and no out-of-order execution the memory ordering is sequential for main memory and strong (sequential) channel 1 for all I/O memory regions.

Coherency and Cachability PMAs

every memory region is coherent and not cacheable, since no cache is implemented.

Idempotency PMAs

All memory regions shall be Idempotent, therefore no checks are applied here.

AHB Master

The AHB master is the interface between the CPU and the system-bus.
It supports the AHB protocol specified by ARMs AMBA 2 specs.
The AHB Master needs a signal to read the date and address that is present at its inputs, as well as specification whether it is a load or store.
After receiving said information, the AHB Master starts the transfer and pulls a signal to HIGH, as soon as it is finished. If it's a load, the data is put on the data output.
The AHB master shall not be able to perform bursts.
The following picture shows a simple AHB data transfer:

Figure 3-3 shows a read transfer with two wait states.

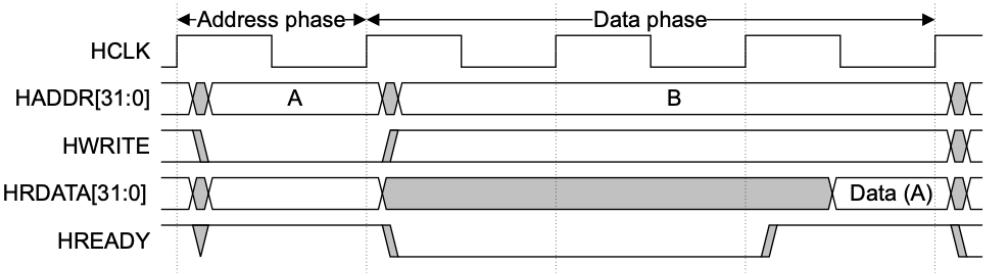


Figure 3-3 Read transfer with two wait states

Figure 3-4 shows a write transfer with one wait state.

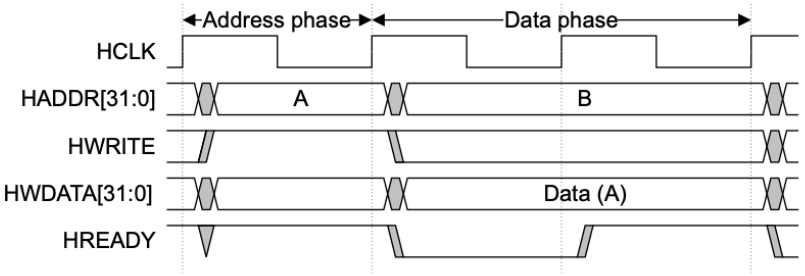


Figure 3-4 Write transfer with one wait state

AHB slave & memory mapped CSRs

The AHB slave & memory mapped CSR unit is comprised of an AHB Slave and some CSRs. The CSRs can be addressed via the memory space, addresses can be specified by setting a generic in the VHDL code. A basic address is predefined for each CSR.

The AHB slave is in charge of accepting AHB transfers and reading/writing the CSRs.

Memory Mapped CSRs:

register	address	access	width	description
msip	0x0200_0000	R/W	32-bit	hold software interrupt pending bit
mtimecmp	0x0200_4000	R/W	32-bit	hold time compare value (lower 32 bit)
mtimecmph	0x0200_4004	R/W	32-bit	hold time compare value (upper 32 bit
mtime	0x0200_BFF8	R/W	32-bit	hold time (lower 32 bit)
mtimeh	0x0200_BFFB	R/W	32-bit	hold time (upper 32 bit)