Azadi MASS Document

Chapter # 1

**Brq Basics**

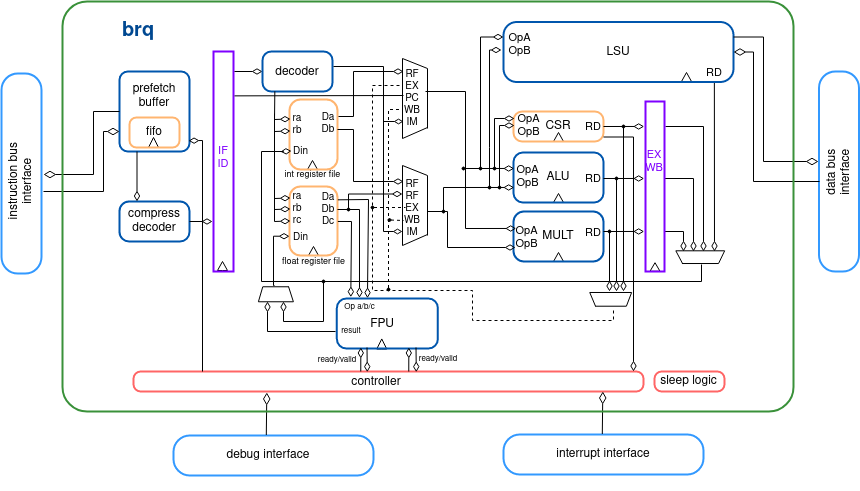
1. **Background**

Brq is 3 stage pipelined core based on ibex, it supports RV32IMFC standard extensions of RISC-V where “I” is the base ISA,”M” is for multiplication and division instructions, “F” is for single precision floating point operations and “C” is for compressed instructions. It supports machine and user mode.

This processor can be used for lower end IoT devices, as this is designed in a way that it can utilize low power.

1. **Brq Overview**

Brq core has three stage pipelining (IFU, [IDU EXU LSU], WBU), in the fetch unit we have a prefetch buffer which contains a 32x3 FIFO, the main purpose of this buffer is to handle the uncompressed instructions. Decode, execute and load store unit is behaving as a single pipeline stage, and write back stage is added to break the data further to improve the frequency.



Chapter # 2

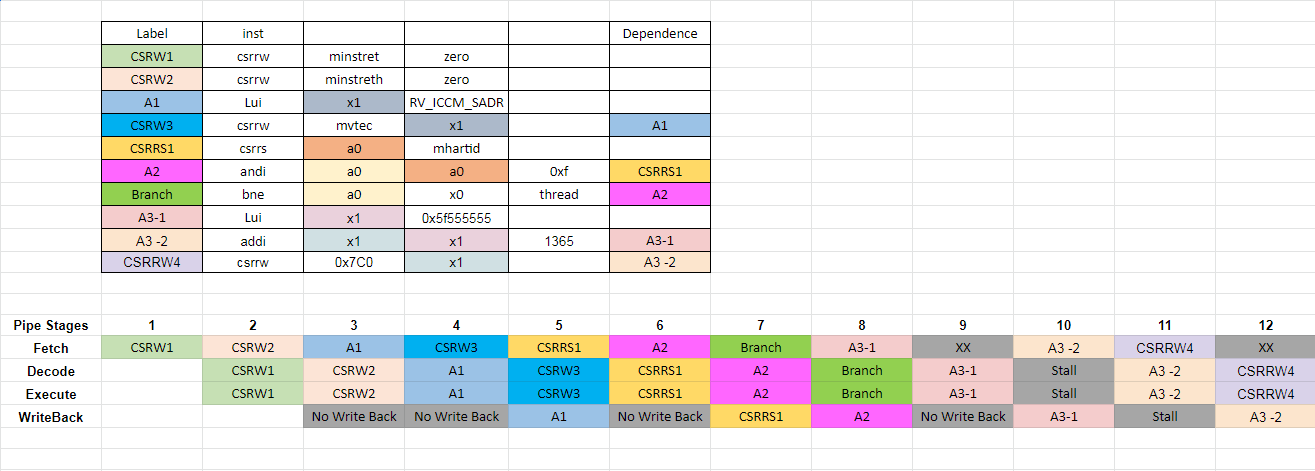
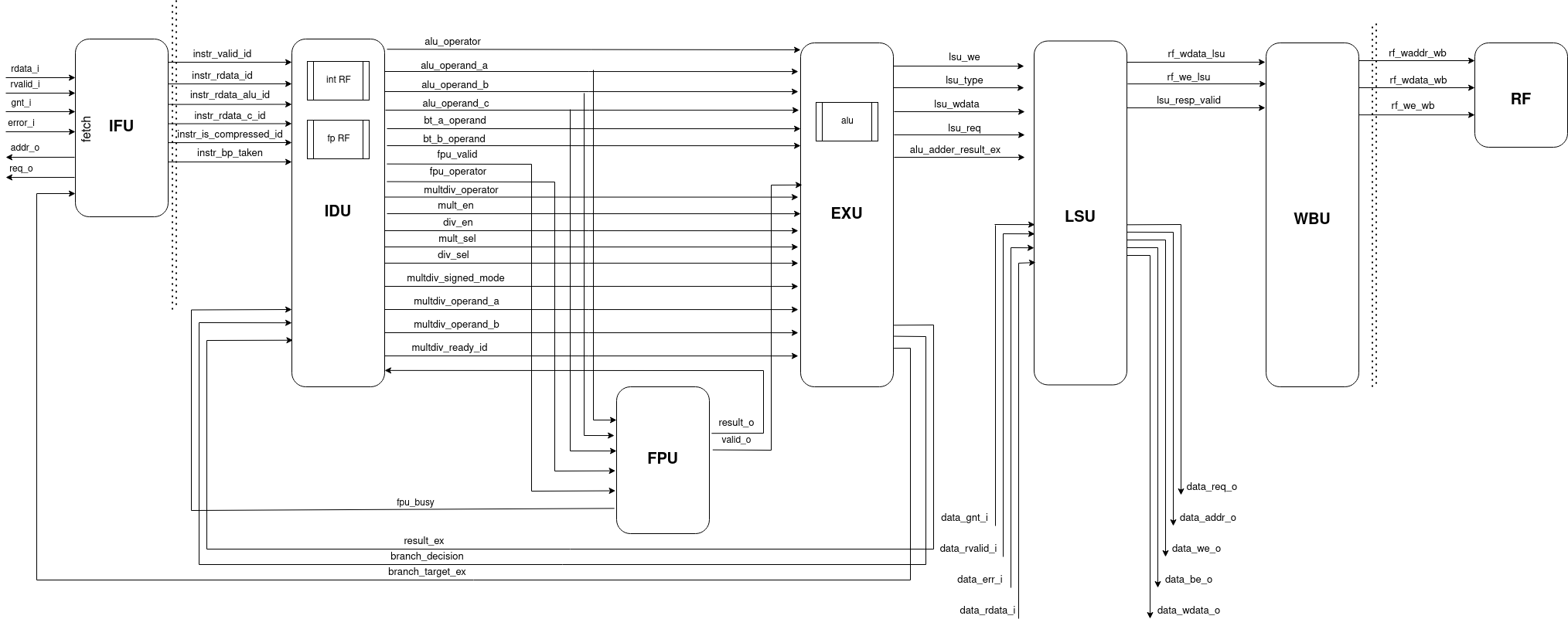
**Introduction to BRQ Core Specification**

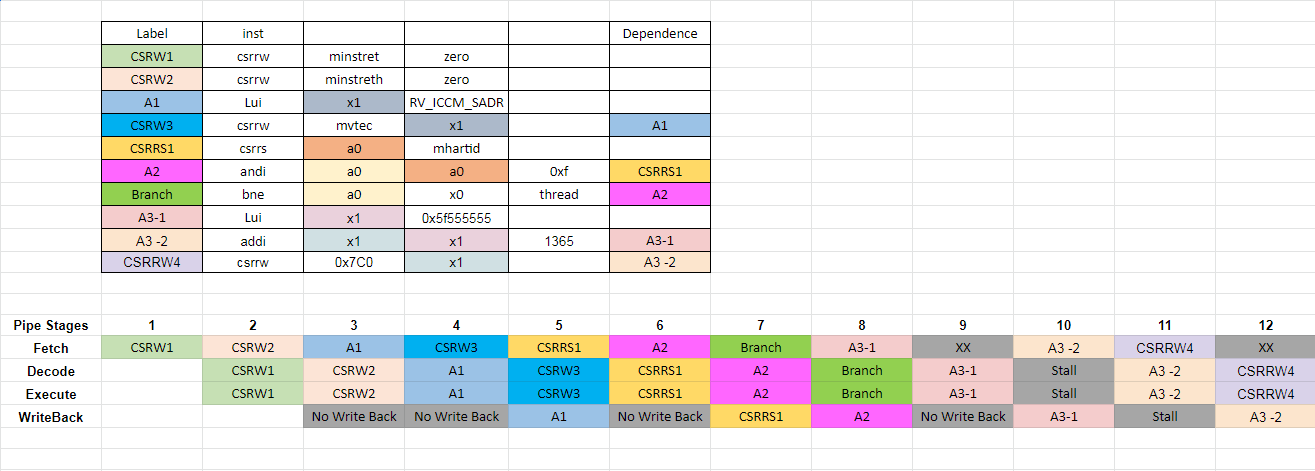
1. **Introduction**

This document is the microarchitecture specification for the brq core.

The primary audience for this document is:

* RISC-V 32 Specification
* 3 stage integer pipeline
* Single thread
* Floating Point Single precision IEEE 754
* Variable floating point pipeline stages
* RISC-V debug
* Interrupts
* Machine and User mode

1. **Integer Pipeline**



Chapter # 3

**Core components**

All instructions require three cycles minimum to pass down the pipeline. One cycle in the ‘Fetch’ stage, one in the ‘Decode and Execute’ stage and one in the ‘Write Back’ stage. Not all instructions can complete its execution in the ‘Decode and Execute’ stage in one cycle so will stall there until they complete. This means the maximum IPC (Instructions per Cycle) brqrv\_eb0 can achieve is one when multi-cycle instructions are not used.

Here are the top level parameters of the core.

| **S.No** | **Parameter Name** | **Type/Range** | **Default** | **Description** |
| --- | --- | --- | --- | --- |
| 1 | PMPEnable | Bit | 0 | Enable PMP support |
| 2 | PMPGranularity | Integer (0…31) | 0 | Minimum granularity of PMP address  matching |
| 3 | PMPNumRegions | Integer (1…16) | 4 | Number implemented PMP regions (ignored  if PMPEnable == 0) |
| 4 | MHPMCounterNum | Integer (0…10) | 0 | Number of performance monitor event  counters |
| 5 | MHPMCounterWidth | Integer (64…1) | 40 | Bit width of performance monitor event  counters |
| 6 | RV32E | Bit | 0 | RV32E mode enable (16 integer registers  only) |
| 7 | RV32M | Integer | RV32MFast | M(ultiply) extension select: “RV32MNone”: No M-extension “RV32MSlow”: Slow multi- cycle multiplier, iterative divider  “RV32MFast”: 3-4 cycle multiplier, iterative divider  “RV32MSingleCycle”: 1-2 cycle multiplier, iterative divider |
| 8 | RV32B | Integer | RV32BNone | B(itmanipulation) extension select: “RV32BNone”: No B-extension  “RV32BBalanced”: Sub-extensions Zbb, Zbs,  Zbf and Zbt  “RV32Full”: All sub-extensions |
| 9 | RegFile | Integer | RegFileFF | Register file implementation select:  “RegFileFF”: Generic flip-flop-based register file  “RegFileFPGA”: Register file for FPGA targets  “RegFileLatch”: Latch-based register file for ASIC targets |
| 10 | BranchTargetALU | Bit | 0 | Enables branch target ALU removing a stall  cycle from taken branches |
| 11 | WritebackStage | Bit | 0 | Enables third pipeline stage (writeback)  improving performance of loads and stores |
| 12 | ICache | Bit | 0 | Enable instruction cache instead of prefetch  buffer |
| 13 | ICacheECC | Bit | 0 | Enable SECDED ECC protection in ICache (if  ICache == 1) |
| 14 | BranchPrediction | Bit | 0 | Enable Static branch prediction |
| 15 | SecureBuraq | Bit | 0 | Enable various additional features targeting secure code execution.  **Note:** SecureBuraq == 1’b1 and RV32M ==  RV32MNone is an illegal combination. |
| 16 | DbgTriggerEn | Bit | 0 | Enable debug trigger support (one trigger  only) |
| 17 | DmHaltAddr | Integer | 0x1A110800 | Address to jump to when entering Debug  Mode |
| 18 | DmExceptionAddr | Integer | 0x1A110808 | Address to jump to when an exception  occurs while in Debug Mode |

In the default configuration, brqrv\_eb0 core is configure as:

* + 1. Ten performance monitor event counters with 32 bits width.
    2. 1-2 cycle multiplier, iterative divider.
    3. Generic flip-flop-based register file.
    4. Branch Target ALU.
    5. Write back Stage.
    6. Debug Trigger support.

This chapter will cover the core components more in detail.

1. Instruction Fetch unit

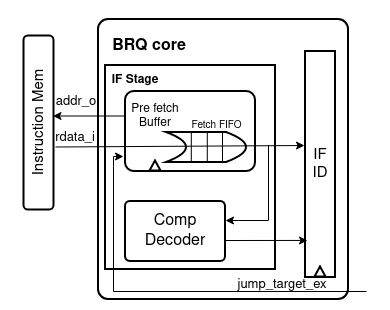
The Instruction Fetch (IF) stage of the core is able to supply one instruction to the Instruction-Decode (ID) stage per cycle if the instruction cache or the instruction memory is able to serve one instruction per cycle.

Instructions are fetched into a prefetch buffer for optimal performance and timing closure reasons. This buffer simply fetches instructions linearly until it is full. The instructions themselves are stored along with the Program Counter (PC) they came from in the fetch FIFO. The fetch FIFO has a feedthrough path so when empty, a new instruction entering the FIFO is immediately made available on the FIFO output. A localparam DEPTH gives a configurable depth which is set to 3 by default.

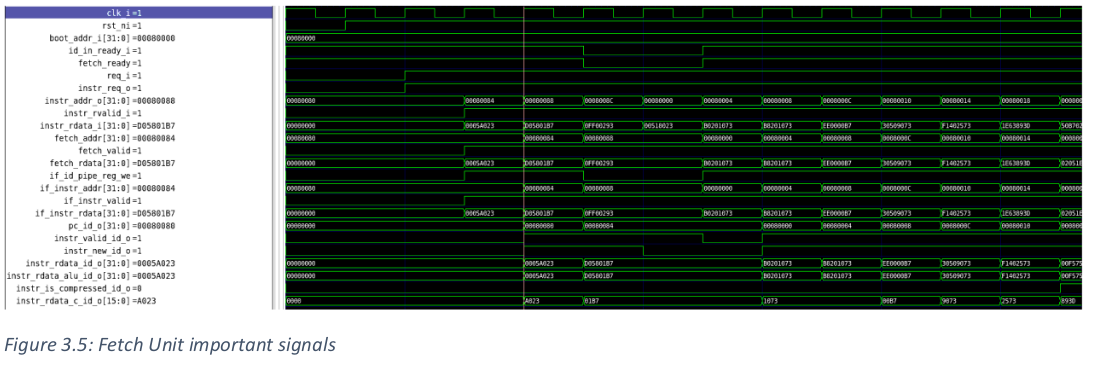
The top-level of the instruction fetch controls the prefetch buffer (in particular flushing it on branches/jumps/exception and beginning prefetching from the appropriate new PC) and supplies new instructions to the ID/EX stage along with their PC. Compressed instructions are expanded by the IF stage so the decoder can always deal with uncompressed instructions (the ID stage still receives the compressed instruction for placing into mtval on an illegal instruction exception).

If Ibex has been configured with an instruction cache (parameter ICache == 1), then the prefetch buffer is replaced by the icache module ([Instruction Cache](https://ibex-core.readthedocs.io/en/latest/03_reference/icache.html#icache)). The interfaces of the icache module are the same as the prefetch buffer with two additions. Firstly, a signal to enable the cache which is driven from a custom CSR. Secondly a signal to flush the cache which is set every time a fence.i instruction is executed.

This unit is responsible for fetching the instruction from instruction memory and sending it to the instruction decoder for decoding. The fetch unit has a prefetch buffer that contains a 32x3 FIFO main used to handle compressed instructions. A compressed decoder which is used to decode the compressed instructions, it converts 16 bit instruction into 32 bit instruction. This conversion saves the hardware cost otherwise there should be a separate decoder for compressed instructions which results in more hardware.



If we have an instruction other than compressed it will bypass the fifo and goes directly to the IF ID pipeline register. The default fetching mechanism is PC+4, it will change once we have taken branches, unconditional jumps or interrupts.



1. Instruction Decode unit
2. Instruction Execute unit
3. Instruction Memory unit
4. Instruction Write Back unit