Function **step()** is the main entry point, it executes one ISS step :

- For an untimed model (PV wrapper) one step corresponds to one instruction.
- For a timed model (CABA wrapper or TLM-T wrapper), one step corresponds to one cycle.

API

inline void reset()

This function resets all registers defining the processor internal state.

inline bool isBusy()

This function is only used by timed wrappers (CABA & TLM-T). In RISC processors, most instructions have a visible latency of one cycle. But some instructions (such as multiplication or division) can have a visible latency longer than one cycle. This function is called by the CABA and TLM-T wrappers before executing one step : If the processor is busy, the wrapper calls the **nullStep()** function. If the processor is available, the wrapper may call the **step()** function to execute one instruction.

inline void step()

This function executes one instruction. All processor internal registers can be modified.

inline void nullStep(int ncycles = 1)

This function performs one internal step of a long instruction.

- `ncycles`: number of cycles to pass with nothing to do, defaults to 1

inline void getInstructionRequest (bool & req , uint32_t & address)

This function is used by the wrappers to obtain from the ISS the instruction request parameters.

- `req`: whether there is a request
- `address`: address of instruction to fetch, must be 4-byte aligned

inline void getDataRequest (bool &req , enum DataAccessType & type, uint32_t & address, uint32_t & wdata)

This function is used by the wrapper to obtain from the ISS the data request parameters.

- `req`: whether there is a request
- `type`: access type, see below
- `address`: address of data access
- `wdata`: data to store, only meaningful for write access types

Type is one of:

```
enum DataAccessType {
    READ_WORD,   // Read Word
    READ_HALF,   // Read Half
    READ_BYTE,   // Read Byte
    LINE_INVAL,  // Cache Line Invalidate
    WRITE_WORD,  // Write Word
    WRITE_HALF,  // Write Half
    WRITE_BYTE,  // Write Byte
    STORE_COND,  // Store Conditional Word
    READ_LINKED, // Load Linked Word
}
```

inline void setInstruction (bool error, uint32_t ins)

This function is used by the wrapper to transmit to the ISS.

- `error`: whether there was an error
- `ins`: instruction for asked address

inline void setDataResponse (bool error, uint32_t rdata)

This function is used by the wrapper to transmit to the ISS, the response to the data request.

- `error`: whether there was an error
- `rdata`: data for asked memory region, only meaningful if access is a read

In any case, this function must reset the ISS data request.

inline void setWriteBerr ()

This function is used by the wrapper to signal asynchronous bus errors, in case of a write acces, that is non blocking for the processor.

inline void setIrq (uint32_t irq)

This function is used by the wrapper to signal the current values of the interrupt lines (as a bitfield) on each cycle.

For each processor, the number of hardware interrupt lines must be defined by the ISS static variable **n_irq**, and is limited to 32.