Contents

Multiple process execution in time division mode.

Process State Object.

Execution of multiple processes is supported by the structure of information called Process State Object – PSO. PSO is used to support the operation of the messaging system and is used in memory allocation functions. PSO is used to store a number of unique process parameters and to store the context of internal registers when the process is not active. PSO uses a separate object in the RAM that can’t be segmented. The information constituting the PSO is located at the object's zero offset.

31	16	15	0	
Regular messages queue				
System messages queue				
Table of imported procedures				
Table of exported procedures				
Contexts stack				+64
Contexts stack pointer				+60
Contexts stack limit				+56
Contexts stack offset				+52
Regular messages queue write pointer		Regular messages queue read pointer		+48
Not used		Regular messages queue length		+44
Regular messages queue offset				+40
System messages queue write pointer		System messages queue read pointer		+36
Not used		System messages queue length		+32
System messages queue offset				+28
Items count of the table of imported procedures				+24
Offset of the table of imported procedures				+20
Items count of the table of exported procedures				+16
Offset of the table of exported procedures				+12
Remained object count				+8
Remained free memory				+4
Not used		Process Timer Base Value [15:0]		+0

The first 64 bytes of the PSO have fixed assignments. Contexts Stack, Table export procedure, Table, import procedure and message queues is a variable length block, may be placed in any order.

Process timer base value. The time of the process activity, expressed by the ticks of the system timer. If it is 0, the timer is not used and the process is limited in some other way. The context controller extracts the value of the process timer base value from the PSO and places it in the PTR register before starting the process. This action is performed only in the mode of cyclic context switching. When you switch to message handlers and interrupts, the timer value is not used.

Remained free memory. It is a value indicating how much more memory can be requested by the process for its needs. The parameter is used by the memory allocation system to monitor the free memory consumption of the process. The value

decreases when a new memory block is allocated and is incremented when the process releases the block of memory. The size is expressed in 32-byte blocks.

Remained object count. The counter of the number of objects that the process can create. The parameter is intended to limit the process in the query of memory blocks by the number of objects. Decreases by 1 when the process requests allocation of the next block of memory and increases by 1, when the process releases the block of memory. The allocation of a new block of memory is blocked if the value of the "remained object count" is 0.

Offset of the table of exported procedures. Indicates where in PSO the table of exported procedures begins.

Items count of the table of exported procedures. It is used to verify the correctness of the procedure index when sending a message to the process to which the PSO belongs.

Offset of the table of imported procedures. Indicates where in PSO the table of exported procedures begins.

Items count of the table of imported procedures. It is used to verify the correctness of the procedure index when sending a message from the process to which the PSO belongs.

System messages queue offset. Indicates the location in the PSO of the buffer of the system messages queue.

System messages queue length. Parameter specifies the maximum number of messages in the system message queue.

System messages queue read pointer. A pointer to the first message in the system message queue.

System messages queue write pointer. Indicates the position in which the next system message will be placed.

Regular messages queue offset. Indicates the location in the PSO of the buffer of the regular messages queue.

Regular messages queue length. Parameter specifies the maximum number of messages in the regular message queue.

Regular messages queue read pointer. A pointer to the first message in the regular message queue.

Regular messages queue write pointer. Indicates the position in which the next regular message will be placed.

Contexts stack offset. Defines location of the contexts stack buffer.

Contexts stack limit. Context stack length, in bytes.

Contexts stack pointer. The context stack expands (contexts stack pointer increased) up when creating the next context frame and shrinks down when the used frame is closed (contexts stack pointer decreased).

Contexts stack. It contains the contexts frames. First frame located from zero offset in the contexts stack buffer.

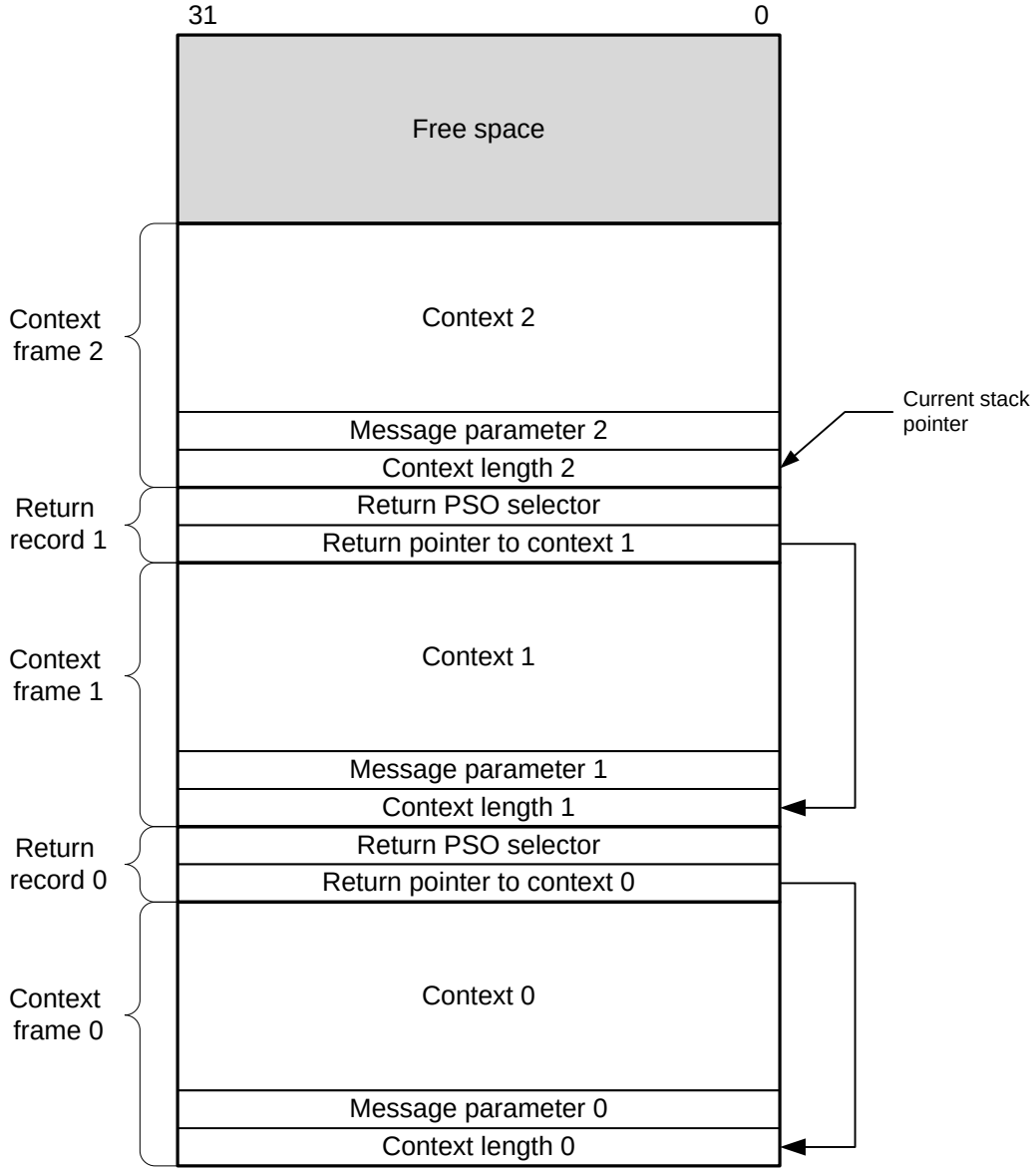
Table of exported procedures. The buffer contains pointers to the message processing procedures that are made available for calling by other processes.

Table of imported procedures. The buffer contains pointers to the message processing procedures that the process itself imports from other processes.

System messages queue. This buffer is used to store incoming system messages.

Regular messages queue. This buffer is used to store incoming regular messages.

Contexts stack.



The context stack contains context frames separated by return records.

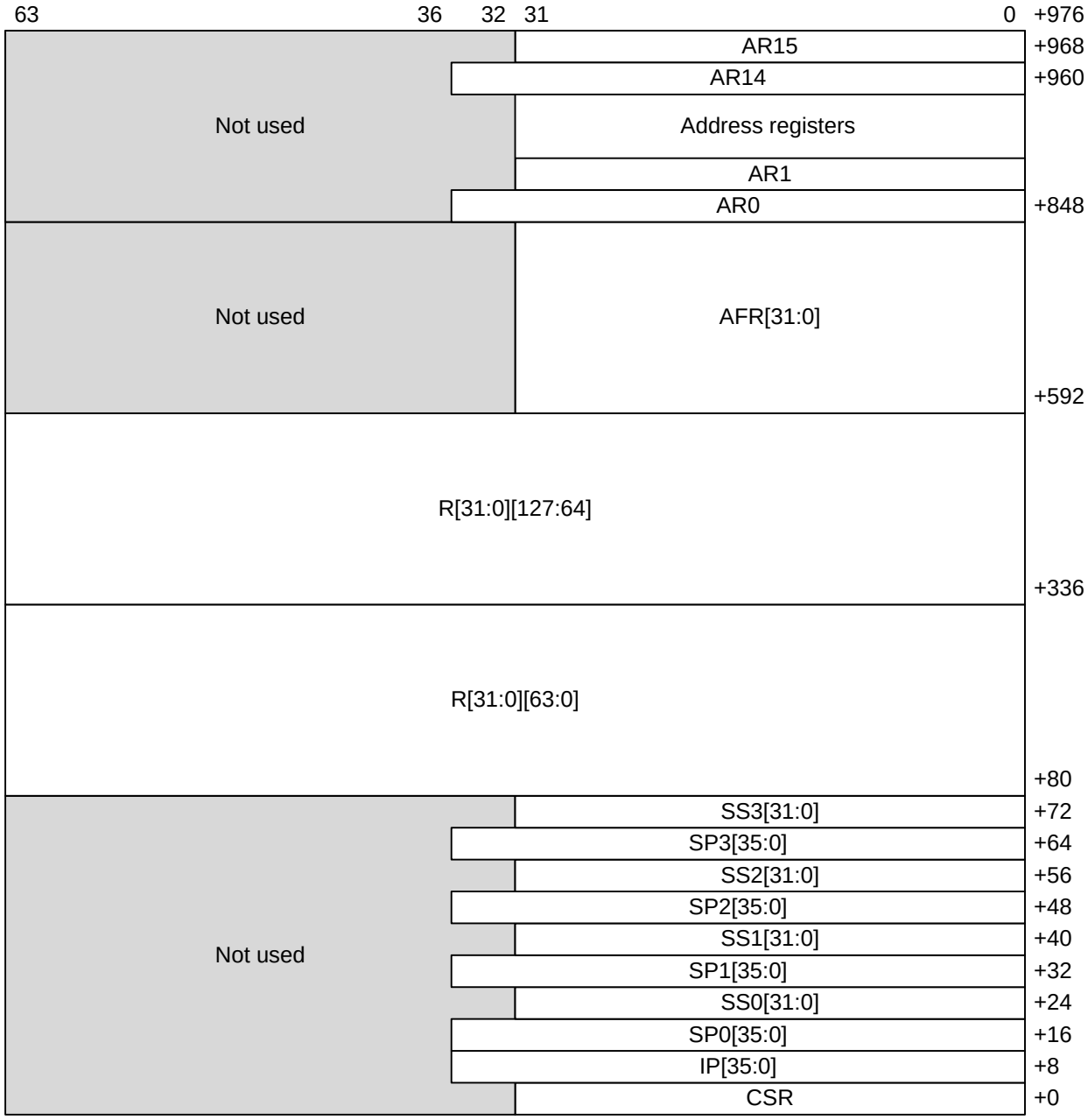
Context frames are used to save the context of the process in times when the process is not being executed by the core. The context frame contains the length of the context, the message parameter, and the actual core context. The context length is used when creating a new frame to determine the location of the return pointer entry. The length of the context is provided in order to make it possible in the future to use reconfigurable cores with different context lengths depending on the core configuration. The message parameter is stored in a context frame so that it can later be extracted with the GETPAR instruction.

The return record contains two values: a return to the previous frame and a PSO return selector. The PSO return selector can point to the current PSO or to another, in the event that any process is interrupted by the interrupt processing. The return pointer is used to make it possible in the future to implement a reconfigurable

core with a variable length of contexts, depending on the configuration. The return pointer is used when the current context frame is closed by the ENDMSG instruction to reset the context stack pointer to the previous frame of the system stack.

The size of the buffer for the stack of contexts is recommended to be chosen based on the possibility of saving at least 5 contexts. The main process loop (context frame 0) can be interrupted by processing a regular message (context frame 1). A regular message can be interrupted by the system message (context frame 2). The system message can be interrupted by a hardware interrupt (context frame 3). A hardware interrupt can be interrupted by processing a security violation (context frame 4).

Process context.

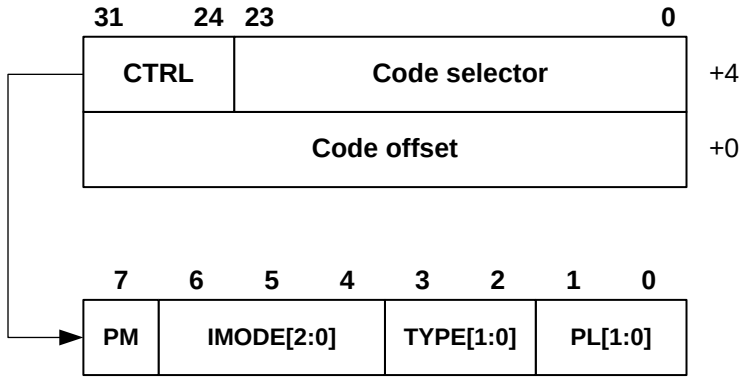


The length of the core context CoreOne32V0 is fixed - 976 bytes. Context includes:

- CSR register content;
- The IP instruction pointer determines the position of the first instruction from which the code execution will start when the process is activated;
- The four sets of program stack pointers are offsets and selectors for the four privilege levels. When the context load occurs, the image of registers AR14 and AR15 is ignored, and one of the pairs SS[3:0]:SP[3:0] is loaded into the registers themselves. Which pair will be used is determined depending on the CPL field from the CSR register. When the context is saved, the corresponding CPL position is updated with the program stack pointer.
- A set of general-purpose registers R[31:0], bits [63:0] and bits [127:64];
- A set of flag registers AFR[31:0];
- A set of address registers AR[15:0].

Table of exported procedures.

The table contains the descriptors of the message processing procedures that are available for calling from other processes.



The descriptor includes a selector and an offset that define the entry point to the message processing procedure. CPU Number is not included into selector. Higher byte set to 0 when selector will be loaded into AR13 register.

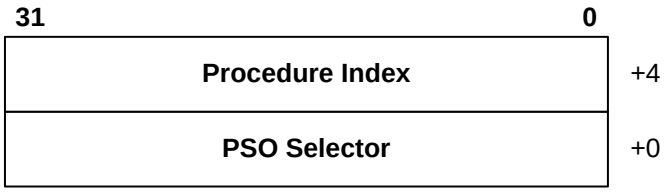
The control byte that defines the mode of message processing, the mode of the privilege level of the handler and the mode of forming the Task ID for the handler. Control byte structure:

- PL is the privilege level used to control access to the procedure descriptor. If the PL of the process calling the message handler is numerically greater than the procedure descriptor PL, access to the entry point is blocked. PL is not checked if the source of the call is a hardware interrupt or a violation of the protection system;
- TYPE. The type of the handler. 0 - interrupt handler, 1 - procedure, 2 - system message, 3 - regular message.
- IMODE. Interrupt blocking mode. Bit 0 - blocks hardware interrupts, bit 1 blocks cyclic switching of processes, bit 2 blocks the call of messages.

- PM. A bit that specifies the mode for setting the properties of the message handler. When PM = 0, the CPL and TaskID of the message handler are set by the values from the descriptor of the code object to which the control will be transferred. PM = 1 - instructs to use CPL and TaskID passed from the process that caused the message handler. This bit is not used if a hardware interrupt or security violation is being processed.

Table of imported procedures.

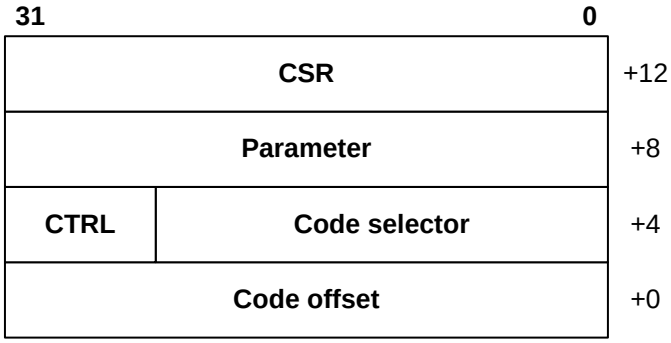
The table contains the descriptors of the imported procedures. Each descriptor contains 2 values: the process selector PSO and the index of the exported procedure in the export procedure table in the specified PSO.



Using these two values, the processor retrieves the handle of the message handler in the table of exported procedures. The PSO selector can reference an object in another processor if the high byte is non-zero and is not equal to the current processor index. In this case, the message is sent to another processor.

Message queues (system and regular).

Any message is always placed in the process message queue, regardless of whether it can be processed immediately or not. A message consisting of four double words is placed in the message queue.

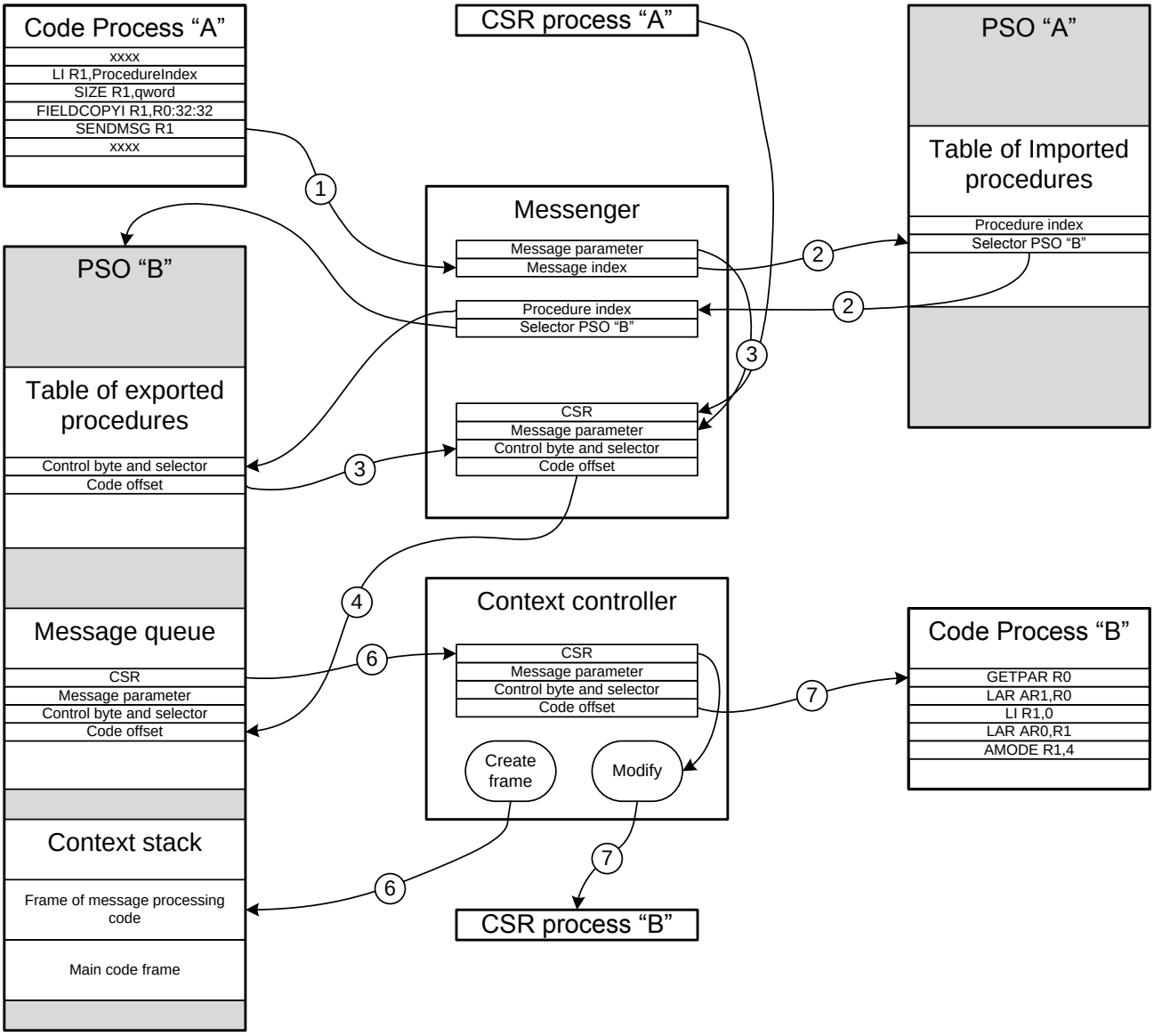


The record includes:

- The offset of the entry point to the handler code and the code object selector;
- The control byte copied from the exported procedures table;
- A 32-bit message parameter. This parameter is passed by the SENDMSG command and is written to the top of the context stack and can be extracted into the message processing routine by the GETPAR instruction.
- The contents of the CSR state register of the process that sent the message, on the basis of which a new CSR state will be generated for the called message processing procedure.

Message or procedure transfer process.

The SENDMSG instruction causes the message to be sent to another process or an immediate procedure call. Algorithms for performing both actions almost completely coincide, except that the procedure call parameters are not set in the process message queue, but are applied immediately switching the core to the execution of the procedural code.



1. Instruction SENDMSG is coming to the messenger unit.
2. The messenger looks at the table of imported procedures in the current PSO using the index specified in the SENDMSG instruction and receives the PSO of the receiving message process and the message handler index in the exported procedures table.
3. Using the obtained PSO selector and index, the messenger retrieves the descriptor of the message handler from the table of exported procedures of

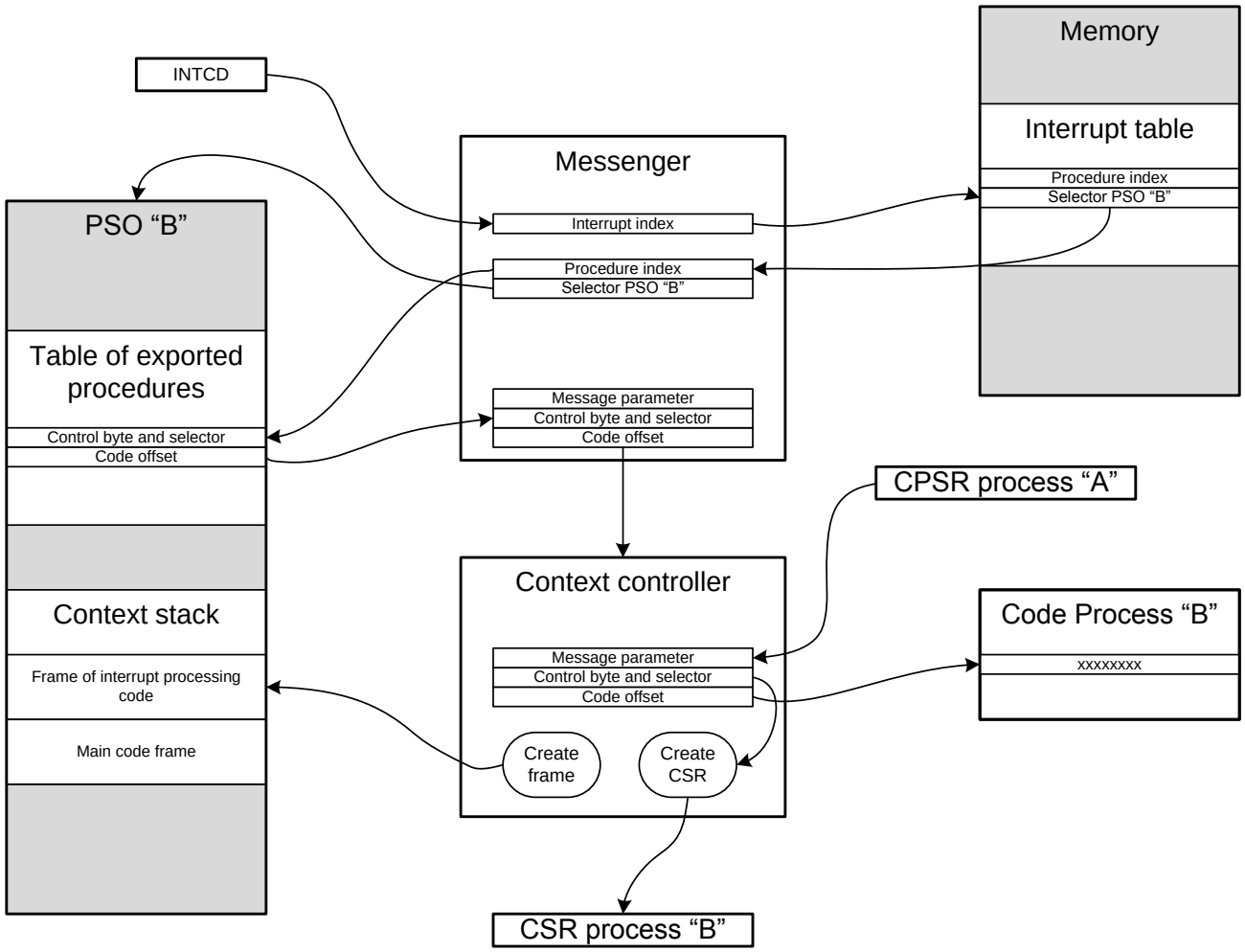
process B. Performs a check on the accessibility of the process “B” message handler from process “A” and generates a message.

4. The message is placed in the message queue of process “B”. This concludes the work of the messenger.
5. If the CSR of process “B” allows message processing, the context controller checks the message queue for process “B” in the following cases:
    - The resumption of the work of process “B” by the mechanism of cyclic switching of processes;
    - In the case where the processes “B” and “A” are the same process;
    - In case the control byte from the table of exported procedures of process “B” defines the message handler as a procedure that must be executed immediately;
    - In case the "B" process has completed the processing of the previous message, procedure or hardware interrupt.
6. If a message is detected in the queue or if there is an immediate request to start the procedure from the messenger, the context controller creates a new frame in the process “B” context stack. This frame will be used to save the context of process “B” in the event that the message handler or procedure is interrupted, for example, by a hardware interrupt or by a procedure for cyclic switching processes. The message parameter is placed in the created context frame and subsequently it can be obtained by the GETPAR command by the message handler.
7. The context controller generates a new CSR register value, sets the instruction pointer to the entry point to the message processing routine, and starts the core to execute the message or procedure handler code.

Message parameter may be a simple 32-bit value but typically an object selector is used. One process takes a memory block, fills block with some parameters and datas and sends an object selector of this memory block to the another process as message parameter.

Interrupt processing.

The INTCR register contains an object selector in which a table is placed that is identical to the table of imported procedures of any process. The number of records in the table is also indicated in the INTCR register.



By processing the hardware interrupt, the messenger reads the 16-bit interrupt ID code from the INTCD bus, indexes the interrupt table, extracting the PSO selector and the index of the exported procedure from it.

Three entry points to the interrupt table have a special purpose. A zero entry point is never used, since the messenger ignores interrupts with a null identifier, treating them as interrupts from uninitialized peripheral equipment. The entry point with index 1 is used to specify the security violation handler. The entry point with index 2 is used to specify the BKPT instruction handler.

The messenger generates a request to the context controller to immediately stop the core and switch it to interrupt processing. The control byte, the PSO selector of the interrupt handler, the code object selector and the interrupt handler offset are passed to the context controller.

The context controller creates a new frame in the context stack and starts the interrupt routine. If the interrupt is caused by the BKPT command, then when the frame is created in the context stack, the controller sets the process selector PSO as the message parameter. This is done so that the debugging interrupt handler can determine which process caused the interrupt.

Checking the message queues.

System messages queue state always checking first and regular messages is not processed while system queue holds at least one message.

Queue checking:

1. When the ENDMSG instruction is executed and message processing is not prohibited in the process to which the return is performed.
2. When a message is addressed to a process that is currently active and message processing is enabled.
3. When the cyclic process switching system activates the process, the context controller checks the status of the message queues of the activated process if message processing is enabled.