Change Log

| | | |
|---|---|---|
| 8.68 | 2024-02-07 | – *rix*/*rcd* abstraction: no more *aperr* if frame attributes are too small |
| | | – added panic case: *alc*/*alcg*/*alct*/*dalc frame* if *frame* = null or *frame* does not refer to a frame |
| | | – system registers now consistently prefixed with *x* (*ctsr xx* = *dz*, cfsr *dx* = *xz*) |
| | 2024-02-02 | – *flshic* renamed to *clric*, *flshbc* renamed to *clrbc* |
| | | – added *clric*, *clrbc*, *flshdc*, *flshac* without arguments to invalidate/flush entire cache |
| | | – *clr* and *flsh* instructions with cache line argument still available, but deprecated |
| | | – *ccc* renamed to *clrcc* |
| | | – reg0/alu08 and reg0/lsu09 renamed to reg0/sys08 and reg0/sys09, re-encoded |
| | 2023-12-05 | – the return of *alcb*: *cbp* re-replaced by *alcb* (no more need for single line flush or force load) |
| 8.67 | 2023-04-18 | – clarified description of *qdth*, *qtdw*, *qdtd* |
| | 2023-03-23 | – added "*trace mode state*"-bit in *x_status* |
| | | – name of all system registers now consistently start with *x*... (instead of *s*...) |
| | 2023-03-17 | – added *x_break_count* register (gc coprocessor interface temorarily removed) |
| 8.66 | 2023-02-06 | – trace mode and breakpoints added |
| | | – system registers reorganized |
| | | – *pcce* now accepts *null* |
| 8.65 | 2023-01-27 | – syntax of *rindex*/*rcode* abstraction instructions changed to *rst rix*, *sv rix*, *rst rcd*, *sv rcd* |
| | | – *rindex* renamed to *rix*, *rcode* renamed to *rcd* |
| | | – new register aliases that mirror the calling convention |
| | 2022-12-28 | – added *rstrix*, *svrix*, *rstrcd*, *svrcd* to save/restore *rix*/*rcd* to/from frame |
| | | – hidden fields for *rix* and *rcd* if *ri*/*rc* are set, do not contribute to π/δ (!) |
| | | – $\pi_{\mathrm{eff}} = \pi + rc$, $\delta_{\mathrm{eff}} = \delta + 4ri$ |
| | 2022-12-27 | – first draft for 64 Bit extension, div extension (many instructions re-encoded) |
| | | – first draft for compressed instructions |
| | | – *muliu*, *mulis* removed (not yet) |
| 8.64 | 2022-12-20 | – new stack alias instructions |
| | 2022-12-xx | – *alcb* replaced by *cbp* |
| | 2022-11-11 | – new: "pointer move" instructions *lcp* (load and clear pointer) and *scp* (store and clear pointer) |
| | | – removed instructions *lssp* and *lssb* (*lssp* replaced by *lcp*, atomic instructions postponed ("todo")) |
| | 2022-11-02 | – *super code object* renamed to *core object* (pointer register alias *core* for *p31*) |
| | | – *super object* renamed to *root object* (pointer register alias *root* for *p31*). |
| | 2022-09-19 | – new: pointer to super code object now encoded as FFFFFFF8 = – 8 (instead of 0) |
| | | – therefore: super code object can now call routines in other code objects |
| | | – therefore: super code object can now be called by *jlib* (to eventually replace *trap*) |
| | | – attributes of super code object held in system registers *s_super_code_xi*, *s_super_code_cxi* |
| | | – attributes of super code object never stored in memory or loaded from memory |
| | | – *jlib super,ix* used to call routines in the super code object, *super* encoded as *p31 = rcode* |
| | | – 0 < *ix* < *s_super_code_xi* (*ix* = 0 illegal, reserved for reset) |
| | | – word at physical address 0 contains branch instruction to reset/initialization routine |
| | | – *jlib super,dy* illegal |
| | 2022-08-18 | – removed *super* from the pointer register file, pointer/attributes of *super* now in system registers |
| | | – super object accessed by dereferencing *p31 = rcode* (privileged) |
| | | – super object managed manually, not allocated by *alc*, not moved by gc (only scanned) |
| | | – pointer to super object exclusively stored in *s_super = s15* |
| | | – attributes of super object exclusively stored in *s_super_pi*/*s_super_delta = s16*/*s17* |
| | | – pointer to super object will never appear in a pointer register, will never be stored in objects |
| | | – pointer to super object will never be loaded from memory, attributes in memory are don't care |
| 8.63 | 2022-01-17 | – instructions completely re-encoded, format now determined by leading bits (big endian!) |
| | | – merged *slt* and *mult* groups |
| | | – *nop* instruction *add d0,d0,d0* now encoded as 00000000 |
| | | – wider displacements for *bra*, *bsr* |
| | | – wider pi and delta fields for *alc* instructions |
| | | – scale factor re-added |
| | | – *trap* immediate now 10 bits |
| | | – removed instructions: *abs*, *ror*, *rol*, *lssh*, *lssw*, *lssp*, *unchk* (!) |
| | | – added instructions: *exthu* |
| 8.62 | 2020-02-07 | – scale factor removed |

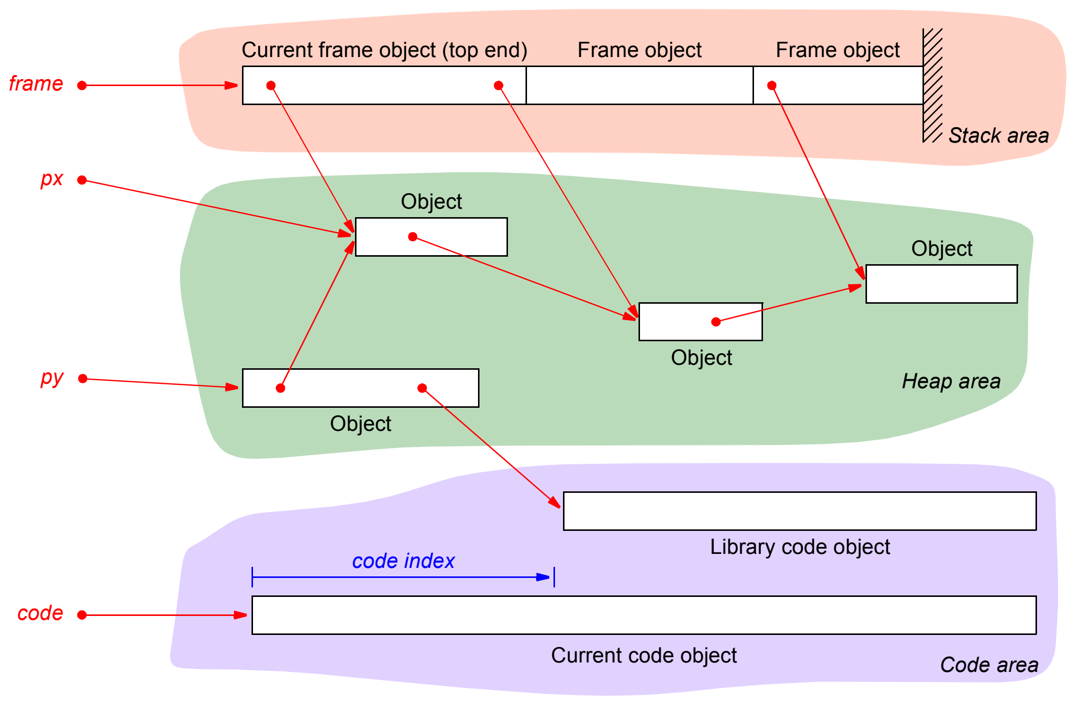
| | | |
|---|---|---|
| 8.61 | 2020-01-24 | – changed instruction encoding, omitted some instructrions (RISC-V inspired) |
| 8.60 | 2020-01-21 | – extended conditional branches to include compare (RISC-V inspired) |

Discussion, Todos

– one word compressed frame attributes?
– root object only accessible from core?
– atomic instructions

## 1 Memory model

### 1.1 User view



The architecture's memory model is object-based: While traditional architectures use addresses to access memory, Objective-RISC identifies a memory location by providing a pointer to an object and an index into that object.

In Objective-RISC, all resources used by programs are represented as objects: Objects are used for instances of structs or records in traditional programming languages as well as for objects in object-oriented languages. Objects are used for arrays, lists and maps. Special stack frame objects are piled on top of each other to form the program stack. Code objects contain the program code, and IO objects provide access to peripherals.

Ordinary objects for program data are created by an allocate instruction that creates an object in the heap area, initializes the created object (with null pointers and zeroes), and writes a pointer to the object to a pointer register. There is no instruction for the deletion of objects. The architecture relies on garbage collection to reclaim memory used by objects that are no longer used. The architecture ensures the integrity of objects and pointers and restricts the set of operations on pointers: Pointers can be stored in objects, loaded from objects, copied in between pointer registers, they can be compared to see whether they refer to the same object, and they can be dereferenced to access the object they refer to. It is not possible to forge pointers, overwrite pointers by non-pointer data or to perform arithmetic operations on pointers.

The program stack consists of special frame objects that are created by a special frame allocate instruction in a memory area designated as the stack area. The top frame object is referred to by a special pointer register called *frame*. The architecture ensures that a subroutine exclusively accesses its own frame object, i.e. the frame object created after entering that subroutine. As a consequence, stack frames cannot be used for parameter passing. A subroutine manually deallocates its frame object before returning to its caller. For this purpose, the architecture provides a deallocate instruction that can exclusively be used on the top frame object. To prevent dangling references, the architecture protects *frame* and ensures that the frame pointer it is never read or copied. There is only one pointer that refers to a frame object, and that pointer is held by *frame*. Pointers to frame objects are never stored in objects, and pointers to frame objects beneath the top frame do not exist.

Program code is organized in code objects. Code objects are used for application programs and for libraries (or frameworks) and typically contain many subroutines. They exclusively contain program code, they do not contain embedded data and, in particular, they do not contain pointers. It is not possible to access code objects by load or store instructions. Code objects are managed by the operating system in a memory area referred to as the code area.

The *code* register holds a pointer to the current code object, and the *code index* register refers to the instruction within the current code object that is to be executed next. In a manner of speaking, the pair of the *code* (pointer) register and the *code index* (data) register corresponds to the program counter register (PC) found in traditional architectures. The *code* and *code index registers* are not part of the standard pointer and data register files (as the PC register is not part of the standard register file in most traditional architectures).

A standard subroutine call, also referred to as an intra code object call, may jump to any index within the current code object. The call instruction saves the code index of the instruction immediately following the call instruction to the *rix* register (return index). A return instruction then uses the value in *rix* to leave the subroutine and to return to the caller.

A code object may call code in other code objects, referred to as library code objects. For this purpose, the application program needs a pointer to the corresponding library. It is not possible to call a subroutine in a library without a pointer to that library. To call a subroutine in a code object other than the current code object, the architecture provides a library call instruction (more precisely: a library entry call instruction), also referred to as an inter code object call. Library entry calls must not only save the code index to *rix* (like an intra code object call), but also the pointer to the current code object. The *rcd* register (return code) is provided for that purpose. A library return instruction (more precisely: a library exit instruction) then uses the values in *rcd* and *rix* to return to the caller.

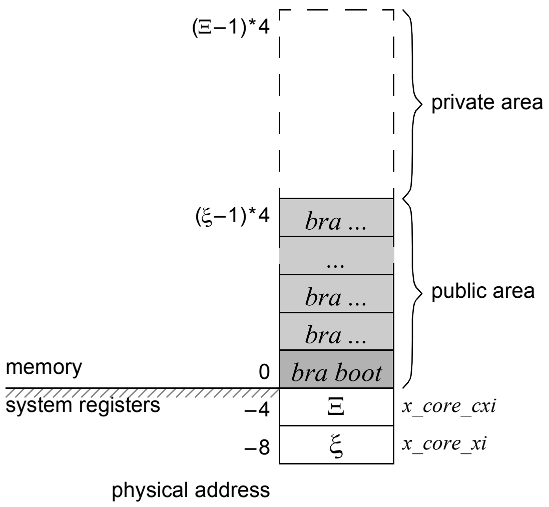
## 1.2 Supervisor view

### 1.2.1 The core object

The *core object* is a code object that contains the bootstrap code and exception handlers for faults, interrupts and traps. In contrast to ordinary code objects, its attributes are not stored in memory, but are held in system registers. This way, the processor does not need to access memory (or the attribute cache) when it switches to the *core object* to initiate exception handling.

The *core object* is located at the beginning of the physical address space, but it does not start at memory address 0. If it did, pointers to the *core object* could not be distinguished from the *null* pointer and it would not be possible for the *core object* to call code in other code objects because the return code object pointer would be *null*. Similarly, it would not be possible to call the *core object* from other code objects since the target code object pointer would be *null*. Therefore, the *core object* starts at address –8 = $FFFFFFF8.

Like any other code object, the *core object* has a public area whose size is described by its $\xi$ attribute. In contrast to other code objects, however, index 0 may not be called as it is reserved for a branch instruction to the bootstrap code. Also in contrast to other code objects, a caller does not require a pointer to the *core object* to call a routine in the *core object*.



### 1.2.2 The root object

The *root object* is used by the *core object* (and other privileged code objects). It can only be accessed in supervisor mode. In a manner of speaking, the *root object* acts as the context object for the *core object*.

The *root object* is manually allocated by supervisor code apart from the heap area, it is not created by an *allocate* instruction. The supervisor determines the location and size of the *root object* by configuring the corresponding system registers *x_root*, *x_root_pi* and *x_root_delta*. Supervisor code accesses the *root object* by dereferencing *p31*. However, it is not possible to read the pointer to the *root object* by reading *p31*, and it is not possible to overwrite the pointer to the *root object* by writing to *p31*. The starting address of the *root object* is exclusively held in *x_root*. Apart from that, there are no pointers to the *root object*, i.e. pointers to the *root object* may never appear in a pointer register or in memory.

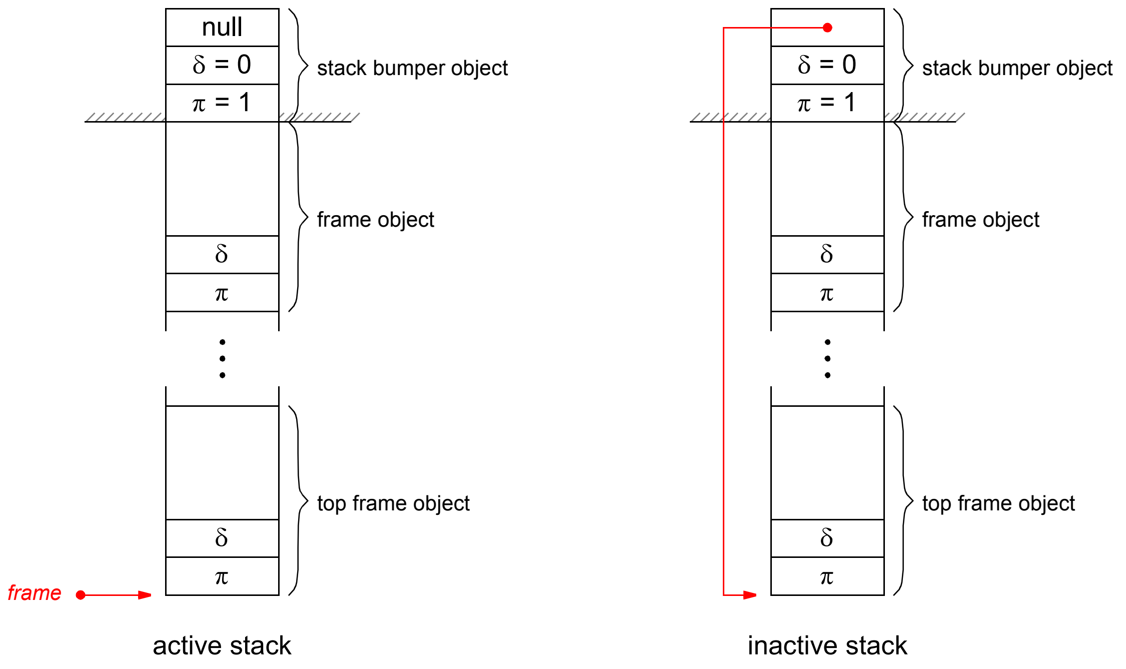
The root set for a garbage collector consists of the *root object* and the pointer register file. The garbage collector will scan the *root object* for pointers and will update them accordingly, but it will never move the *root object*.

### 1.2.3 The stack bumper object

Objective-RISC supports multiple stacks, but only one stack can be active at any given time. The top frame object of the active stack is always referred to by pointer register *p30 = frame*.

If an active stack is empty, *frame* will point to the stack's *stack bumper object*. A *stack bumper object* has the size to acommodate a single pointer, and the value of that pointer is *null* if the corresponding stack is currently active. The *stack bumper* of an inactive stack always contains a pointer to the top frame object.

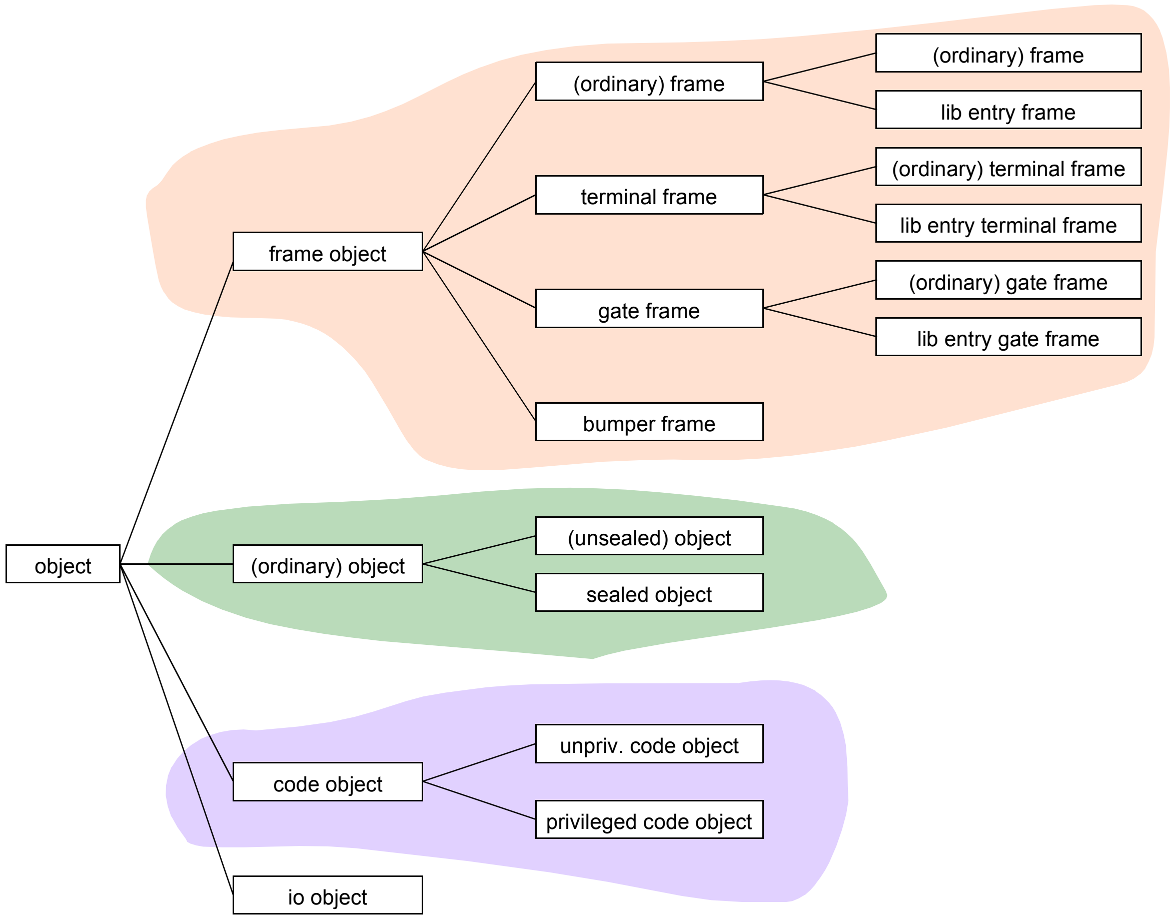
*Stack bumper objects* are created by a privileged *allocate bumper* instruction that takes the physical memory address of the *stack bumper object* to be created and returns a pointer to that *stack bumper object* in *frame*.



1.2.4 Stack management

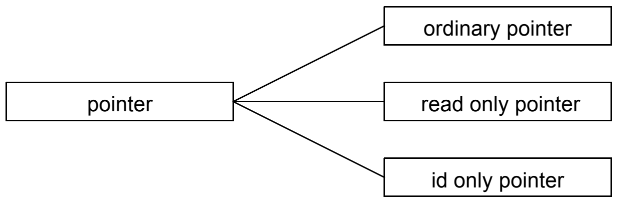
## 1.3 Objects and Pointers

### 1.3.1 Object kinds



Unsealed objects and frame objects can be uninitialized (more precisely: incompletely initialized) if their initialization is suspended by an interrupt. Pointers to uninitialized objects are exclusively managed by the supervisor and may never become visible in user mode.

### 1.3.2 Pointer kinds



Only for ordinary objects (sealed or unsealed)

# 2 Register model

## 2.1 Register model in user mode

| | Pointer registers | | Data registers |
|---|---|---|---|
| p0 | null | d0 | zero |
| p1 | | d1 | |
| p2 | | d2 | |
| … | … | … | … |
| p29 | | d29 | |
| p30 | frame | d30 | |
| p31 | rcd/core | d31 | rix |
| | | | |
| PC | code | | code index |

## 2.2 Register model in supervisor mode

| | Pointer registers | | Data registers |
|---|---|---|---|
| p0 | null | d0 | zero |
| p1 | | d1 | |
| p2 | | d2 | |
| … | … | … | … |
| p29 | | d29 | |
| p30 | frame | d30 | |
| p31 | rcd/core/root | d31 | rix |
| | | | |
| PC | code | | code index |

| | System registers | |
|---|---|---|
| x0 | x_version | |
| x1 | x_status | System state and configuration |
| x2 | x_options | |
| x3 | x_core_xi | Core attributes |
| x4 | x_core_cxi | |
| x5 | x_root | |
| x6 | x_root_pi | Root object |
| x7 | x_root_delta | |
| x8 | x_exc_code_pnt | |
| x9 | x_exc_code_xi | |
| x10 | x_exc_code_cxi | |
| x11 | x_exc_code_index | Exception state |
| x12 | x_exc_status | |
| x13 | x_exc_alc_state | |
| x14 | x_exc_no | |
| x15 | x_frame_lim | Stack |
| x16 | x_intv | |
| x17 | x_faultv | Exception vectors |
| x18 | (x_trapv) | |
| x19 | x_tracev | |
| x20 | x_break_code | |
| x21 | x_break_index | Breakpoint |
| x22 | x_break_count | |
| x23 | | |
| x24 | x_fspc_start | |
| x25 | x_fspc_end | |
| x26 | x_tspc_start | |
| x27 | x_tspc_end | Heap configuration |
| x28 | x_alc_addr | |
| x29 | x_alc_lim | |
| x30 | | |
| x31 | | |

## System register details

| | x_status | x_options |
|---|---|---|
| 0 | interrupt enable | local branch target cache enable |
| 1 | trace mode enable | local return stack enable |
| 2 | trace mode state | global branch target cache enable |
| 3 | | global return stack enable |
| 4 | | conditional branch prediction enable |
| 5 | | |
| 6 | | |
| 7 | | |
| 8 | | |
| 9 | | |
| 10 | | |
| 11 | | |
| 12 | | |
| 13 | | |
| 14 | | |
| 15 | | |
| 16 | | |
| 17 | | |
| 18 | | |
| 19 | | |
| 20 | | |
| 21 | | |
| 22 | | |
| 23 | | |
| 24 | | |
| 25 | | |
| 26 | | |
| 27 | | |
| 28 | | |
| 29 | | |
| 30 | | |
| 31 | | |

## 2.3 Register aliases and calling convention

| | Pointer registers | | Data registers |
|---|---|---|---|
| p0 | null | d0 | zero |
| p1 | h5 | d1 | t5 |
| p2 | h6 | d2 | t6 |
| p3 | b3 | d3 | a3 |
| p4 | b4 | d4 | a4 |
| p5 | b5 | d5 | a5 |
| p6 | b6 | d6 | a6 |
| p7 | b7 | d7 | a7 |
| p8 | **q0** | d8 | **s0** |
| p9 | **q1** | d9 | **s1** |
| p10 | b0 | d10 | a0 |
| p11 | b1 | d11 | a1 |
| p12 | b2 | d12 | a2 |
| p13 | h0 | d13 | t0 |
| p14 | h1 | d14 | t1 |
| p15 | h2 | d15 | t2 |
| p16 | h3 | d16 | t3 |
| p17 | h4 | d17 | t4 |
| p18 | **q2** | d18 | **s2** |
| p19 | **q3** | d19 | **s3** |
| p20 | **q4** | d20 | **s4** |
| p21 | **q5** | d21 | **s5** |
| p22 | **q6** | d22 | **s6** |
| p23 | **q7** | d23 | **s7** |
| p24 | **q8** | d24 | **s8** |
| p25 | **q9** | d25 | **s9** |
| p26 | **q10** | d26 | **s10** |
| p27 | **q11** | d27 | **s11** |
| p28 | **cnst** | d28 | **s12** |
| p29 | **ctxt** | d29 | **s13** |
| p30 | **frame** | d30 | **s14** |
| p31 | **rcd/core** | d31 | **rix** |
| | | | |
| PC | code | | code index |

8 argument registers a0 – a7; b0 – b7

7 temporary registers t0 – t6; h0 – h6

16 saved registers s0 – s14, rix; q0 – q11, cnst, ctxt, frame, rcd

## 2.4 Pointer register *p31*

Objective-RISC uses *p31* for three different physical registers that refer to three different objects: The *return code object*, the *core object* and the *root object*. Correspondingly, there are three symbolic aliases for register *p31*, namely *rcd*, *core* and *root*. The physical register *p31* along with its attributes is used for *rcd*. The pointer to the *core object* is a constant, the *core object*'s attributes are held in the system registers *x_core_xi* and *x_core_cxi*. The pointer to the *root object* and the attributes of the *root object* are held in the system registers *x_root*, *x_root_pi* and *x_root_delta*.

| | | | |
|---|---|---|---|
| rcd | rcd | π(rcd) | δ(rcd) |
| core | −8 | x_core_xi | x_core_cxi |
| root | x_root | x_root_pi | x_root_delta |

It is always clear from the context which of the three physical registers is actually meant by *p31*. As a general rule, *rcd* is used whenever *p31* is read or written, *root* is used whenever *p31* is dereferenced, and *core* is used if *p31* is used as the target code object of a *jlib* instruction.The following table shows all instructions whose pointer operand (or operands) may be *p31*, and which of the three physical registers is used in each case (unprivileged instructions are printed in bold).

| Instruction | *px* = *p31* refers to | *py* = *p31* refers to |
|---|---|---|
| `cpp   py = px` | *rcd* | *rcd* |
| `ccp   px = dy` | *rcd* | |
| **`cpfc  rcd`** | *rcd* | |
| **`check rcd`** | *rcd* | |
| `lbu/s dy = px[ix12]` | | |
| `lhu/s dy = px[ix12]` | | |
| `lw    dy = px[ix12]` | *root* | |
| `lp    py = px[ix12]` | | *rcd* |
| `lcp   py = px[ix12]` | | *rcd* |
| `sb    px[ix12] = dy` | | |
| `sh    px[ix12] = dy` | | |
| `sw    px[ix12] = dy` | *root* | |
| `sp    px[ix12] = py` | | *rcd* |
| `scp   px[ix12] = py` | | *rcd* |
| `lbu/s dy = px[dz*s3+ud4]` | | |
| `lhu/s dy = px[dz*s3+ud4]` | | |
| `lw    dy = px[dz*s3+ud4]` | *root* | |
| `lp    py = px[dz*s3+ud4]` | | *rcd* |
| `lcp   py = px[dz*s3+ud4]` | | *rcd* |
| `sb    px[dz*s3+ud4] = dy` | | |
| `sh    px[dz*s3+ud4] = dy` | | |
| `sw    px[dz*s3+ud4] = dy` | *root* | |
| `sp    px[dz*s3+ud4] = py` | | *rcd* |
| `scp   px[dz*s3+ud4] = py` | | *rcd* |
| `qpi   dy = px` | | |
| `qpir  dy = px` | | |
| `qdtr  dy = px` | | |
| `qptr  dy = px` | *root* | |
| `qdtb  dy = px` | | |
| `qdth  dy = px` | | |
| `qdtw  dy = px` | | |
| `qdtd  dy = px` | | |
| **`jlib  px,ix20`** | *core* | |
| **`rtlb`** | *rcd* | |

# 3 Instruction set

## 3.1 Overview

### 3.1.1 Unprivileged instructions (user mode)

| | | | |
|---|---|---|---|
| `addi dx = dy,ui12`<br>`subi dx = dy,ui12` | `add dx = dy,dz`<br>`sub dx = dy,dz` | Add<br>Subtract | |
| `andi dx = dy,ui12`<br>`andni dx = dy,ui12`<br>`ori dx = dy,ui12`<br>`xori dx = dy,ui12` | `and dx = dy,dz`<br>`andn dx = dy,dz`<br>`or dx = dy,dz`<br>`xor dx = dy,dz` | And<br>And not<br>Or<br>Xor | |
| `sltiu dx = dy,ui12`<br>`sltis dx = dy,si12` | `sltu dx = dy,dz`<br>`slts dx = dy,dz` | Set if less than unsigned<br>Set if less than signed | |
| `muliu dx = dy,ui12`<br>`mulis dx = dy,si12` | <br><br>`mul dx = dy,dz`<br>`mulhu dx = dy,dz`<br>`mulhs dx = dy,dz`<br>`mulhsu dx = dy,dz` | Multiply unsigned<br>Multiply signed<br>Multiply<br>Multiply higher unsigned<br>Multiply higher signed<br>Multiply higher signed unsigned | Data processing instructions (ALU) |
| `srli dx = dy,ui5`<br>`srai dx = dy,ui5`<br>`slli dx = dy,ui5` | `srl dx = dy,dz`<br>`sra dx = dy,dz`<br>`sll dx = dy,dz` | Shift right logical<br>Shift right arithmetic<br>Shift left logical | |
| | `exthu dx = dz`<br>`extbs dx = dz`<br>`exths dx = dz` | Extend half word unsigned<br>Extend byte signed<br>Extend half word signed | |
| | `not dx = dz` | Not | |
| | `clz dx = dz` | Count leading zeros | |
| `lui dx = ui20` | | Load upper immediate | |
| `alc px = dy,δ15`<br>`alc px = π15,dz`<br>`alc px = π9,δ11`<br>`alc frame = π9,δ11`<br>`alct frame = π9,δ11`<br>`alcg frame = π9,δ11` | `alc px = dy,dz` | Allocate object<br>Allocate object<br>Allocate object<br>Allocate stack frame<br>Allocate terminal stack frame<br>Allocate gate stack frame | Pointer generating instructions (PGU) |
| | `dalc frame` | Deallocate stack frame | |
| | `cpp py = px` | Copy pointer | |
| | `gcp px` | Get context pointer | |
| | `cpfc rcd` | Copy the code pointer to rcd | |
| `lbu/s dy = px[ix12]`<br>`lhu/s dy = px[ix12]`<br>`lw dy = px[ix12]`<br>`lp py = px[ix12]`<br>`lcp py = px[ix12]` | `lbu/s dy = px[dz*s3+ud4]`<br>`lhu/s dy = px[dz*s3+ud4]`<br>`lw dy = px[dz*s3+ud4]`<br>`lp py = px[dz*s3+ud4]`<br>`lcp py = px[dz*s3+ud4]` | Load byte unsiged/signed<br>Load half word unsigned/signed<br>Load word<br>Load pointer<br>Load and clear pointer | |
| `sb px[ix12] = dy`<br>`sh px[ix12] = dy`<br>`sw px[ix12] = dy`<br>`sp px[ix12] = py`<br>`scp px[ix12] = py` | `sb px[dz*s3+ud4] = dy`<br>`sh px[dz*s3+ud4] = dy`<br>`sw px[dz*s3+ud4] = dy`<br>`sp px[dz*s3+ud4] = py`<br>`scp px[dz*s3+ud4] = py` | Store byte<br>Store half word<br>Store word<br>Store pointer<br>Store and clear pointer | Load and store instructions (LSU) |
| `rst rix`<br>`rst rcd` | | Restore return index<br>Restore return code pointer | |
| `sv rix`<br>`sv rcd` | | Save return index<br>Save return code pointer | |
| | `qpi dy = px`<br>`qdtb dy = px`<br>`qdth dy = px`<br>`qdtw dy = px`<br>`qdtd dy = px` | Query π attribute<br>Query δ attribute in number of bytes<br>Query δ attribute in number of half words<br>Query δ attribute in number of words<br>Query δ attribute in number of double words | Attribute instructions (ATU) |
| | `nchk px` | Null check | |
| `beqp px,py,sd12`<br>`bnep px,py,sd12`<br>`beq dy,dz,sd12`<br>`bne dy,dz,sd12`<br>`bgeu dy,dz,sd12`<br>`bges dy,dz,sd12`<br>`bltu dy,dz,sd12`<br>`blts dy,dz,sd12` | | Branch if pointers are equal<br>Branch if pointers are not equal<br>Branch if equal<br>Branch if not equal<br>Branch if greater or equal unsigned<br>Branch if greater or equal signed<br>Branch if less than unsigned<br>Branch if less than signed | |
| `bra sd25` | `jmp dy` | Branch/jump unconditionally | Branch instructions (BPU) |
| `bsr sd25` | `jsr dy`<br>`rts` | Branch/jump to subroutine<br>Return from subroutine | |
| `jlib px,ix20` | `jlib px,dy`<br>`rtlb` | Jump to library entry routine<br>Return from library entry routine | |
| | `check rcd` | Check *rcd* | |
| `trap ui10` | | System call | |

### 3.1.2 Privileged instructions (supervisor mode)

| | | | |
|---|---|---|---|
| | `ctsr   xx = dz`<br>`cfsr   dx = xz` | Copy to system register<br>Copy from system register | ALU |
| | `crop   py = px`<br>`cidp   py = px`<br>`rpr    py = px` | Create read only pointer<br>Create id only pointer<br>Restore pointer rights | |
| | `seal   px`<br>`unsl   px` | Seal object<br>Unseal object | PGU |
| | `alcb   frame = dy` | Allocate stack bumper | |
| | `ciop   px = dy,dz`<br>`ccp    px = dy` | Create io pointer<br>Create code pointer | |
| | `flshic dz`<br>`flshdc dz`<br>`flshac dz`<br>`flshbc dz` | Flush instruction cache line<br>Flush data cache line<br>Flush attribute cache line<br>Flush branch target cache line | LSU |
| | `pcce   dz,px`<br>`rcce   dz`<br>`ccc` | Put context cache entry<br>Remove context cache entry<br>Clear context cache | |
| | `qpir   dy = px`<br>`qdtr   dy = px`<br>`qptr   dy = px` | Query raw $\pi$ attribute<br>Query raw $\delta$ attribute<br>Query raw pointer | ATU |
| | `sync`<br>`rte` | Sync<br>Return from exception | BPU |

### 3.1.3 Privileged versions of otherwise unprivileged instructions (see opcode map notes 1 – 6)

| | | | |
|---|---|---|---|
| 1 | $y = r$ | **`lw     rix = px[ix12]`**<br>**`lp     rcd = px[ix12]`**<br>**`lcp    rcd = px[ix12]`**<br>**`sw     px[ix12] = rix`**<br>**`sp     px[ix12] = rcd`**<br>**`scp    px[ix12] = rcd`** | manage *rix*, *rcd* |
| 2 | $y = r$ | **`lw     rix = px[dz*s3+ud4]`**<br>**`lp     rcd = px[dz*s3+ud4]`**<br>**`lcp    rcd = px[dz*s3+ud4]`**<br>**`sw     px[dz*s3+ud4] = rix`**<br>**`sp     px[dz*s3+ud4] = rcd`**<br>**`scp    px[dz*s3+ud4] = rcd`** | manage *rix*, *rcd* |
| 3 | $y = f$ | **`lp     frame = px[0]`**<br>**`lcp    frame = px[0]`**<br>**`sp     px[0] = frame`**<br>**`scp    px[0] = frame`** | load or store *frame* from or to a stack bumper, illegal if not ($x \neq f$ and $ix12 = 0$) |
| 4 | $x = 0$ | **`lbu/s  dy = null[ix12]`**<br>**`lhu/s  dy = null[ix12]`**<br>**`lw     dy = null[ix12]`**<br>**`sb     null[ix12] = dy`**<br>**`sh     null[ix12] = dy`**<br>**`sw     null[ix12] = dy`**<br>**`lbu/s  dy = null[dz*s3+ud4]`**<br>**`lhu/s  dy = null[dz*s3+ud4]`**<br>**`lw     dy = null[dz*s3+ud4]`**<br>**`sb     null[dz*s3+ud4] = dy`**<br>**`sh     null[dz*s3+ud4] = dy`**<br>**`sw     null[dz*s3+ud4] = dy`** | *px = null*<br>access to physical memory (linear address space of bytes)<br>no attributes skipped in address calculation<br>index expression not implicitly scaled<br>unaligned memory access raises *sverr* fault |
| 5 | $x = r$ | **`lbu/s  dy = root[ix12]`**<br>**`lhu/s  dy = root[ix12]`**<br>**`lw     dy = root[ix12]`**<br>**`lp     py = root[ix12]`**<br>**`lcp    py = root[ix12]`**<br>**`sb     root[ix12] = dy`**<br>**`sh     root[ix12] = dy`**<br>**`sw     root[ix12] = dy`**<br>**`sp     root[ix12] = py`**<br>**`scp    root[ix12] = py`**<br>**`lbu/s  dy = root[dz*s3+ud4]`**<br>**`lhu/s  dy = root[dz*s3+ud4]`**<br>**`lw     dy = root[dz*s3+ud4]`**<br>**`lp     py = root[dz*s3+ud4]`**<br>**`lcp    py = root[dz*s3+ud4]`**<br>**`sb     root[dz*s3+ud4] = dy`**<br>**`sh     root[dz*s3+ud4] = dy`**<br>**`sw     root[dz*s3+ud4] = dy`**<br>**`sp     root[dz*s3+ud4] = py`**<br>**`scp    root[dz*s3+ud4] = py`** | *px = root (p31)*<br>access to root object |
| 6 | $x = r$<br>$y = r$ | **`cpp    py = rcd`**<br>**`cpp    rcd = px`** | manage *rcd* |

## 3.1.4 Pseudo instructions and aliases

| Pseudo instruction/alias | Implemented as |
|---|---|
| nop | add d0 = d0,d0 |
| cp dx = dy | add dx = dy,d0 |
| li dx = ui12 | add dx = d0,ui12 |
| lni dx = ui12 | sub dx = d0,ui12 |
| clr dx | add dx = d0,d0 |
| inc dx | addi dx = dx,1 |
| dec dx | subi dx = dx,1 |
| extbu dx = dy | andi dx = dy,255 |
| neg dx = dy | sub dx = d0,dy |
| qdt dy = px | qdtb dy = px |
| stack | |
| push ui9,ui9 | alc frame := ui9,ui11 |
| pushg ui9,ui9 | alcg frame := ui9,ui11 |
| pusht ui9,ui9 | alct frame := ui9,ui11 |
| pop | dalc frame |
| sv dx,ui12 | sw frame[ui12] := dx |
| sv px,ui12 | sp frame[ui12] := px |
| rst dx,ui12 | lw dx := frame[ui12] |
| rst px,ui12 | lp px := frame[ui12] |
| swapped operand versions | |
| mulhus dx = dy,dz | mulhsu dx = dz,dy |
| bgts dy,dz,sd12 | blts dz,dy,sd12 |
| bles dy,dz,sd12 | bges dz,dy,sd12 |
| bgtu dy,dz,sd12 | bltu dz,dy,sd12 |
| bleu dy,dz,sd12 | bgeu dz,dy,sd12 |
| two-address versions | |
| addi dx = ui12 | addi dx = dx,ui12 |
| subi dx = ui12 | subi dx = dx,ui12 |
| andi dx = ui12 | andi dx = dx,ui12 |
| andni dx = ui12 | andni dx = dx,ui12 |
| ori dx = ui12 | ori dx = dx,ui12 |
| xori dx = ui12 | xori dx = dx,ui12 |
| muliu dx = ui12 | muliu dx = dx,ui12 |
| mulis dx = si12 | mulis dx = dx,si12 |
| srli dx = ui5 | srli dx = dx,ui5 |
| srai dx = ui5 | srai dx = dx,ui5 |
| slli dx = ui5 | slli dx = dx,ui5 |
| add dx = dz | add dx = dx,dz |
| sub dx = dz | sub dx = dx,dz |
| and dx = dz | and dx = dx,dz |
| andn dx = dz | andn dx = dx,dz |
| or dx = dz | or dx = dx,dz |
| xor dx = dz | xor dx = dx,dz |
| mul dx = dz | mul dx = dx,dz |
| srl dx = dz | srl dx = dx,dz |
| sra dx = dz | sra dx = dx,dz |
| sll dx = dz | sll dx = dx,dz |
| unary versions | |
| seq dx = dy | sltiu dx = dy,1 |
| sne dx = dy | sltu dx = d0,dy |
| slt dx = dy | slts dx = dy,d0 |
| sgt dx = dy | slts dx = d0,dy |
| beqp px,sd12 | beqp px,p0,sd12 |
| bnep px,sd12 | bnep px,p0,sd12 |
| beq dx,sd12 | beq dx,d0,sd12 |
| bne dx,sd12 | bne dx,d0,sd12 |
| ble dx,sd12 | bges d0,dx,sd12 |
| bge dx,sd12 | bges dx,d0,sd12 |
| blt dx,sd12 | blts dx,d0,sd12 |
| bgt dx,sd12 | blts d0,dx,sd12 |

optional p-suffix: cpp, beqp, bnep
optional i-suffix: addi, subi, andi, andni, ori, xori, srli, srai, slli, sltiu, sltis, muliu, mulis
optional s-suffix: extbs, exths, slts, sltis, mulis, mulhs, lbs, lhs, bges, blts, (bgts, bles)

## 3.2 Instruction set details

### 3.2.1 Data processing instructions (ALU)

| | | | |
|---|---|---|---|
| Instruction | `addi  dx = dy,ui12` | Add immediate | *dx* = *dy* + *ui12* |
| | `add   dx = dy,dz` | Add | *dx* = *dy* + *dz* |
| | `subi  dx = dy,ui12` | Subtract immediate | *dx* = *dy* – *ui12* |
| | `sub   dx = dy,dz` | Subtract | *dx* = *dy* – *dz* |
| Description | Add the zero-extended immediate *ui12* to the value in *dy* and store the result in *dx*.<br>Add *dz* to the value in *dy* and store the result in *dx*.<br>Subtract the zero-extended immediate *ui12* from the value in *dy* and store the result in *dx*.<br>Subtract *dz* from the value in *dy* and store the result in *dx*. | | |
| Remarks | Carry and overflow are ignored. | | |

| | | | |
|---|---|---|---|
| Instruction | `andi  dx = dy,ui12` | Bitwise logical AND immediate | *dx* = *dy* & *ui12* |
| | `and   dx = dy,dz` | Bitwise logical AND | *dx* = *dy* & *dz* |
| | `andni dx = dy,ui12` | Bitwise logical AND NOT immediate | *dx* = *dy* & !*ui12* |
| | `andn  dx = dy,dz` | Bitwise logical AND NOT | *dx* = *dy* & !*dz* |
| | `ori   dx = dy,ui12` | Bitwise logical OR immediate | *dx* = *dy* \| *ui12* |
| | `or    dx = dy,dz` | Bitwise logical OR | *dx* = *dy* \| *dz* |
| | `xori  dx = dy,ui12` | Bitwise logical EXCLUSIVE OR immediate | *dx* = *dy* ^ *ui12* |
| | `xor   dx = dy,dz` | Bitwise logical EXCLUSIVE OR | *dx* = *dy* ^ *dz* |
| Description | Perform the respective bitwise logical operation with the zero-extended immediate *ui12* and *dy* and store the result in *dx*.<br>Perform the respective bitwise logical operation with *dy* and *dz* and store the result in *dx*. | | |

| | | | |
|---|---|---|---|
| Instruction | `sltiu dx = dy,ui12` | Set it less than immediate unsigned | *dx* = 1 if *dy* < *ui12* (unsigned), *dx* = 0 otherwise |
| | `sltu  dx = dy,dz` | Set it less than unsigned | *dx* = 1 if *dy* < *dz* (unsigned), *dx* = 0 otherwise |
| | `sltis dx = dy,si12` | Set it less than immediate signed | *dx* = 1 if *dy* < *si12* (signed), *dx* = 0 otherwise |
| | `slts  dx = dy,dz` | Set it less than signed | *dx* = 1 if *dy* < *dz* (signed), *dx* = 0 otherwise |
| Description | Perform an unsigned compare and write 1 to *dx* if *dy* is less than the zero-extended immediate *ui12,* 0 otherwise.<br>Perform an unsigned compare and write 1 to *dx* if *dy* is less than *dz,* 0 otherwise.<br>Perform a signed compare and write 1 to *dx* if *dy* is less than the sign-extended immediate *si12,* 0 otherwise.<br>Perform a signed compare and write 1 to *dx* if *dy* is less than *dz,* 0 otherwise. | | |

| | | | |
|---|---|---|---|
| Instruction | `muliu  dx = dy,ui12` | Multiply immediate unsigned | $dx = (dy * ui12)_{31..0}$ |
| | `mulis  dx = dy,si12` | Multiply immediate signed | $dx = (dy * si12)_{31..0}$ |
| | `mul    dx = dy,dz` | Multiply | $dx = (dy * dz)_{31..0}$ |
| | `mulhu  dx = dy,dz` | Multiply higher unsigned | $dx = (dy_{\text{unsigned}} * dz_{\text{unsigned}})_{63..32}$ |
| | `mulhs  dx = dy,dz` | Multiply higher signed | $dx = (dy_{\text{signed}} * dz_{\text{signed}})_{63..32}$ |
| | `mulhsu dx = dy,dz` | Multiply higher signed unsigned | $dx = (dy_{\text{signed}} * dz_{\text{unsigned}})_{63..32}$ |
| Description | Multiply *dy* with the zero-extended immediate *ui12* and write the lower half of the product to *dx*.<br>Multiply *dy* with the sign-extended immediate *si12* and write the lower half of the product to *dx*.<br>Multiply *dy* with *dz* and write the lower half of the product to *dx*.<br>Multiply unsigned *dy* with unsigend *dz* and write the higher half of the product to *dx*.<br>Multiply signed *dy* with sigend *dz* and write the higher half of the product to *dx*.<br>Multiply signed *dy* with unsigend *dz* and write the higher half of the product to *dx*. | | |

| | | | |
|---|---|---|---|
| Instruction | `srli  dx = dy,ui5` | Shift right logical immediate | *dx* = *dy* >> *ui5* (unsigned) |
| | `srl   dx = dy,dz` | Shift right logical | *dx* = *dy* >> (*dz* & 0x1F) (unsigned) |
| | `srai  dx = dy,ui5` | Shift right arithmetic immediate | *dx* = *dy* >> *ui5* (signed) |
| | `sra   dx = dy,dz` | Shift right arithmetic | *dx* = *dy* >> (*dz* & 0x1F) (signed) |
| | `slli  dx = dy,ui5` | Shift left logical immediate | *dx* = *dy* << *ui5* |
| | `sll   dx = dy,dz` | Shift left logical | *dx* = *dy* << (*dz* & 0x1F) |
| Description | Shift *dy* right by *ui5* positions, shift zeros into the upper bits, and write the result to *dx*.<br>Shift *dy* right by (*dz* & 0x1F) positions, shift zeros into the upper bits, and write the result to *dx*.<br>Shift *dy* right by *ui5* positions, shift the original sign bit into the upper bits, and write the result to *dx*.<br>Shift *dy* right by (*dz* & 0x1F) positions, shift the original sign bit into the upper bits, and write the result to *dx*.<br>Shift *dy* left by *ui5* positions, shift zeros into the lower bits, and write the result to *dx*.<br>Shift *dy* left by (*dz* & 0x1F) positions, shift zeros into the lower bits, and write the result to *dx*. | | |
| Remarks | Carry and overflow are ignored. | | |

| | | | |
|---|---|---|---|
| Instruction | `exthu  dx = dz` | Extend half word unsigned | *dx* = *dz* & 0xFFFF |
| | `extbs  dx = dz` | Extend byte signed | *dx* = *dz* & 0xFF if bit 7 of *dy* = 0,<br>*dx* = *dz* \| 0xFFFFFF00 otherwise |
| | `exths  dx = dz` | Extend half word signed | *dx* = *dz* & 0xFFFF if bit 15 of *dy* = 0,<br>*dx* = *dz* \| 0xFFFF0000 otherwise |
| Description | Copy the lower half of *dz* (bits 15..0) to the lower half of *dx*, fill the higher half of *dx* with zeros.<br>Copy the lowest byte in *dz* (bits 7..0) to the lowest byte of *dx*, fill the rest of *dx* with the sign of the lowest byte in *dz* (bit 7).<br>Copy the lower half of *dz* (bits 15..0) to the lower half of *dx*, fill the higher half of *dx* with the sign of the lower half word in *dz* (bit 15). | | |
| Remarks | The instruction *extbu dx = dz* is provided as a pseudo instruction and implemented as *andi dx = dz*,0xFF. | | |

| | | | |
|---|---|---|---|
| Instruction | `not  dx = dz` | Not | *dx* = !*dz* |
| Description | Perform a bitwise logical *not* operation with *dz* and store the result in *dx*. | | |

| | | | |
|---|---|---|---|
| Instruction | **`clz dx = dz`** | Count leading zeros | *dx* = number of leading zeros in *dy* |
| Description | Count the number of leading zeros in *dz* and write the result to *dx* (*dx* = 0 ... 32). | | |
| Remarks | Used to speed up division in software. | | |

| | | | |
|---|---|---|---|
| Instruction | **`lui dx = ui20`** | Load upper immediate | *dx* = *ui20* << 12 |
| Description | Shift *ui20* left by 12 positions and write the result to *dx* (lower 12 bits of *dx* are all zero). | | |

## 3.2.2 Pointer instructions (PGU)

| | |
|---|---|
| Instruction | `alc px = dy,δ15` Allocate object with $\pi = dy$, $\delta = \delta 15$<br>`alc px = π15,dz` Allocate object with $\pi = \pi 15$, $\delta = dz$<br>`alc px = π9,δ11` Allocate object with $\pi = \pi 9$, $\delta = \delta 11$<br>`alc px = dy,dz` Allocate object with $\pi = dy$, $\delta = dz$ |
| Description | Allocate an ordinary object with $\pi$ pointers and $\delta$ data bytes and return a pointer to the allocated object in *px*. The parameters for $\pi$ and $\delta$ are unsigned, immediate values for $\pi$ and $\delta$ are zero-extended. |
| Faults | *aperr* if $\pi \geq 2^{29}$ or $\delta \geq 2^{31}$<br>*hpovf* if the allocated object would exceed *alc_lim*. |

| | |
|---|---|
| Instruction | `alc frame = π9,δ11` Allocate stack frame object with $\pi = \pi 9$, $\delta = \delta 11$ |
| Description | Allocate an ordinary stack frame object with $\pi$ pointers and $\delta$ data bytes and return a pointer to the allocated object in *frame*. The immediate parameters for $\pi$ and $\delta$ are unsigned and zero-extended. |
| Precodition | any quarantine state (*QC*, *QL* or *QU*) |
| Postcondition | regular state *RC* or *RU*<br>the allocated stack frame is tagged as a library entry stack frame if the instruction is executed in state *QL* |
| Faults | *panic* if *frame* = null or if *frame* does not hold a frame pointer<br>*sterr* if executed in a regular state<br>~~*aperr* if $\delta < 4$~~<br>*stovf* if the allocated stack frame object would exceed *frame_lim*. |
| Remarks | The frame contains a hidden field to save and restore *rix*. |

| | |
|---|---|
| Instruction | `alct frame = π9,δ11` Allocate terminal stack frame with $\pi = \pi 9$, $\delta = \delta 11$ |
| Description | Allocate a terminal stack frame object with $\pi$ pointers and $\delta$ data bytes and return a pointer to the allocated object in *frame*. The immediate parameters for $\pi$ and $\delta$ are unsigned and zero-extended. |
| Precodition | any quarantine state (*QC*, *QL* or *QU*) |
| Postcondition | regular state *RC* or *RU*<br>the allocated stack frame is tagged as a library entry stack frame if the instruction is executed in state *QL* |
| Faults | *panic* if *frame* = null or if *frame* does not hold a frame pointer<br>*sterr* if executed in a regular state<br>*stovf* if the allocated stack frame object would exceed *frame_lim*. |
| Remarks | The frame does not contain hidden fields to save and restore *rix* and *rcd*. |

| | |
|---|---|
| Instruction | `alcg frame = π9,δ11` Allocate gate stack frame with $\pi = \pi 9$, $\delta = \delta 11$ |
| Description | Allocate a gate stack frame object with $\pi$ pointers and $\delta$ data bytes and return a pointer to the allocated object in *frame*. The immediate parameters for $\pi$ and $\delta$ are unsigned and zero-extended. |
| Precodition | any quarantine state (*QC*, *QL* or *QU*) |
| Postcondition | regular state *RC* or *RU*<br>the allocated stack frame is tagged as a library entry stack frame if the instruction is executed state *QL* |
| Faults | *panic* if *frame* = null or if *frame* does not hold a frame pointer<br>*sterr* if executed in a regular state<br>~~*aperr* if $\delta < 4$ or if $\pi < 1$~~<br>*stovf* if the allocated stack frame object would exceed *frame_lim*. |
| Remarks | The frame contains hidden fields to save and restore *rix* and *rcd*. |

| | |
|---|---|
| Instruction | `dalc frame` Deallocate stack frame |
| Description | Deallocate the current stack frame referred to by *frame*. |
| Precodition | regular state *RC* or *RU* |
| Postcondition | quarantine state *QC* (coming from state *RC*)<br>quarantine state *QU* (coming from state *RU*, deallocating a non library entry stack frame)<br>quarantine state *QL* (coming from state *RU*, deallocating a library entry stack frame) |
| Faults | *panic* if *frame* = null or if *frame* does not hold a frame pointer<br>*sterr* if executed in a quarantine state |
| Remarks | Stack underflow (i.e. the deallocation of a stack bumper) is prevented by the stack protection system. |

| | |
|---|---|
| Instruction | `cpp py = px` Copy pointer |
| Description | Copy the pointer stored in *px* to *py* |

| | |
|---|---|
| Instruction | `gcp px` Get context pointer |
| Description | Get the pointer to the context object associated with the current code object |
| Faults | *ccmiss* if the requested pointer is not contained in the context cache |
| Remarks | Required to store the state of library code objects. It is the task of the operation to keep a map of library code objects and their corresponding context objects and to manage the context cache. |

| | |
|---|---|
| Instruction | **`cpfc rcd`** Copy from code pointer to rcd |
| Description | copy the pointer to the current code object to *rcd* |
| Remarks | Required in library objects before calling code in other library objects. |

## 3.2.3 Load and store instructions (LSU)

| | |
|---|---|
| Instruction | `rst rix` |
| Description | Load the return index from the current frame object. |
| Precodition | regular state *RC* or *RU* |
| Faults | *sterr* if in a quarantine state<br>*fram0* if *px* refers to a terminal frame and *dy* = *rix*<br>*fram0* if *px* refers to an ordinary or a gate frame and *dy* ≠ *rix and* index < 4 (load byte), index < 2 (load halfword), index < 1 (load word) |

| | |
|---|---|
| Instruction | `sv rix` |
| Description | Store the return index to the current frame object. |
| Precodition | regular state *RC* or *RU* |
| Faults | *sterr* if in a quarantine state<br>*fram0* if *px* refers to a terminal frame and *dy* = *rix*<br>*fram0* if *px* refers to an ordinary or a gate frame and *dy* ≠ *rix* and index < 4 (store byte), index < 2 (store halfword), index < 1 (store word) |

| | |
|---|---|
| Instruction | `rst rcd` |
| Description | Load the return code pointer from the current frame object. |
| Precodition | *todo* |
| Faults | *todo*<br>*sterr* if *px* = *frame* and in a quarantine state<br>*fram0* if *px* refers to a terminal frame and *dy* = *rix*<br>*fram0* if *px* refers to an ordinary or a gate frame and *dy* ≠ *rix and* index < 4 (load byte), index < 2 (load halfword), index < 1 (load word) |

| | |
|---|---|
| Instruction | `sv rcd` |
| Description | Store the return code pointer to the current frame object. |
| Precodition | *todo* |
| Faults | *todo*<br>*sterr* if *px* = *frame* and in a quarantine state<br>*fram0* if *px* refers to a terminal frame and *dy* = *rix*<br>*fram0* if *px* refers to an ordinary or a gate frame and *dy* ≠ *rix* and index < 4 (store byte), index < 2 (store halfword), index < 1 (store word) |

| | |
|---|---|
| Instruction | `lbu dy = px[ix12]` Load byte unsigned<br>`lbu dy = px[dz*s3+ud4]` Load byte unsigned<br>`lbs dy = px[ix12]` Load byte signed<br>`lbs dy = px[dz*s3+ud4]` Load byte signed<br>`lhu dy = px[ix12]` Load half word unsigned<br>`lhu dy = px[dz*s3+ud4]` Load half word unsigned<br>`lhs dy = px[ix12]` Load half word signed<br>`lhs dy = px[dz*s3+ud4]` Load half word signed<br>`lw  dy = px[ix12]` Load word<br>`lw  dy = px[dz*s3+ud4]` Load word |
| Description | Load a byte, halfword or word from the object referred to by *px*. The index is either given as a zero-extended immediate *ix12* or as an expression *dz***s3*+*ud4* where the index is obtained by first multiplying the value in *dz* with a scale factor *s3* ∈ {1, ..., 8} and then adding an unsigned displacement *u4* ∈ {0, ..., 15} to the product. Halfword and word instructions implicitly scale the index. Depending on the instruction, the loaded byte or halfword is zero-extended or sign-extended. |
| Precodition | if *px* = *frame*, the processor must be in a regular state (*RC* or *RU*) |
| Faults | *privv* if the instruction is a privileged case<br>*drfnu* if *px* = null<br>*drfid* if *px* holds an id pointer<br>*sterr* if *px* = *frame* and in a quarantine state<br>*drfcd* if *px* refers to a code object<br>*ixoob* if the result of the index expression exceeds the size of a word (i.e. if carries occur)<br>*ixoob* if index ≥ δ (load byte), index*2+1 ≥ δ (load halfword), index*4+3 ≥ δ (load word)<br>*fram0* if *px* refers to a terminal frame and *dy* = *rix*<br>*fram0* if *px* refers to an ordinary or a gate frame and *dy* ≠ *rix and* index < 4 (load byte), index < 2 (load halfword), index < 1 (load word) |

| | |
|---|---|
| Instruction | **`sb px[ix12] = dy`** Store byte<br>**`sb px[dz*s3+ud4] = dy`** Store byte<br>**`sh px[ix12] = dy`** Store half word<br>**`sh px[dz*s3+ud4] = dy`** Store half word<br>**`sw px[ix12] = dy`** Store word<br>**`sw px[dz*s3+ud4] = dy`** Store word |
| Description | Store a byte, halfword or word to the object referred to by *px*. The index is either given as a zero-extended immediate *ix12* or as an expression *dz***s3*+*ud4* where the index is obtained by first multiplying the value in *dz* with a scale factor *s3* ∈ {1, ..., 8} and then adding an unsigned displacement *u4* ∈ {0, ..., 15} to the product. Halfword and word instructions implicitly scale the index. |
| Precodition | if *px* = *frame*, the processor must be in a regular state (*RC* or *RU*) |
| Faults | *privv* if the instruction is a privileged case<br>*drfnu* if *px* contains the null value<br>*wrptv* if *px* holds a read only pointer<br>*drfid* if *px* holds an id pointer<br>*sterr* if *px* = *frame* and in a quarantine state<br>*drfcd* if *px* refers to a code object<br>*sealv* if *px* refers to a sealed object<br>*ixoob* if the result of the index expression exceeds the size of a word (i.e. if carries occur)<br>*ixoob* if index ≥ δ (store byte), index*2+1 ≥ δ (store halfword), index*4+3 ≥ δ (store word)<br>*fram0* if *px* refers to a terminal frame and *dy* = *rix*<br>*fram0* if *px* refers to an ordinary or a gate frame and *dy* ≠ *rix* and index < 4 (store byte), index < 2 (store halfword), index < 1 (store word) |

| | |
|---|---|
| Instruction | **`lp py = px[ix12]`** Load pointer<br>**`lp py = px[dz*s3+ud4]`** Load pointer |
| Description | Load a pointer from the object referred to by *px* to *py*. The index is either given as a zero-extended immediate *ix12* or as an expression *dz***s3*+*ud4* where the index is obtained by first multiplying the value in *dz* with a scale factor *s3* ∈ {1, ..., 8} and then adding an unsigned displacement *u4* ∈ {0, ..., 15} to the product. |
| Precodition | if *px* = *frame*, the processor must be in a regular state (*RC* or *RU*) |
| Faults | *privv* if the instruction is a privileged case<br>*drfnu* if *px* contains the null value<br>*drfid* if *px* holds an id pointer<br>*sterr* if *px* = *frame* and in a quarantine state<br>*drfcd* if *px* refers to a code object<br>*ixoob* if the result of the index expression exceeds the size of a word (i.e. if carries occur)<br>*ixoob* if index ≥ π<br>*fram0* if *px* refers to a terminal or an ordinary frame and *py* = *rcd*<br>*fram0* if *px* refers to a gate frame and *py* ≠ *rcd* and index = 0 |

| | |
|---|---|
| Instruction | **`lcp py = px[ix12]`** Load and clear pointer<br>**`lcp py = px[dz*s3+ud4]`** Load and clear pointer |
| Description | Load a pointer from the object referred to by *px* to *py* and then overwrite the pointer in the object with *null*. The index is either given as a zero-extended immediate *ix12* or as an expression *dz***s3*+*ud4* where the index is obtained by first multiplying the value in *dz* with a scale factor *s3* ∈ {1, ..., 8} and then adding an unsigned displacement *u4* ∈ {0, ..., 15} to the product. |
| Precodition | if *px* = *frame*, the processor must be in a regular state (*RC* or *RU*) |
| Faults | *privv* if the instruction is a privileged case<br>*drfnu* if *px* contains the null value<br>*drfid* if *px* holds an id pointer<br>*sterr* if *px* = *frame* and in a quarantine state<br>*drfcd* if *px* refers to a code object<br>*ixoob* if the result of the index expression exceeds the size of a word (i.e. if carries occur)<br>*ixoob* if index ≥ π<br>*fram0* if *px* refers to a terminal or an ordinary frame and *py* = *rcd*<br>*fram0* if *px* refers to a gate frame and *py* ≠ *rcd* and index = 0 |

| | |
|---|---|
| Instruction | **`sp px[ix12] = py`** Store pointer<br>**`sp px[dz*s3+ud4] = py`** Store pointer |
| Description | Store a pointer in *py* to the object referred to by *px*. The index is either given as a zero-extended immediate *ix12* or as an expression *dz***s3*+*ud4* where the index is obtained by first multiplying the value in *dz* with a scale factor *s3* ∈ {1, ..., 8} and then adding an unsigned displacement *u4* ∈ {0, ..., 15} to the product. |
| Precodition | if *px* = *frame*, the processor must be in a regular state (*RC* or *RU*) |
| Faults | *privv* if the instruction is a privileged case<br>*drfnu* if *px* contains the null value<br>*wrptv* if *px* holds a read only pointer<br>*drfid* if *px* holds an id pointer<br>*sterr* if *px* = *frame* and in a quarantine state<br>*drfcd* if *px* refers to a code object<br>*sealv* if *px* refers to a sealed object<br>*ixoob* if the result of the index expression exceeds the size of a word (i.e. if carries occur)<br>*ixoob* if index ≥ π<br>*fram0* if *px* refers to a terminal or an ordinary frame and *py* = *rcd*<br>*fram0* if *px* refers to a gate frame and *py* ≠ *rcd* and index = 0 |

| | |
|---|---|
| Instruction | **`scp px[ix12] = py`** Store and clear pointer<br>**`scp px[dz*s3+ud4] = py`** Store and clear pointer |
| Description | Store the pointer in *py* to the object referred to by *px* and then overwrite *py* with *null*. The index is either given as a zero-extended immediate *ix12* or as an expression *dz*s3+ud4* where the index is obtained by first multiplying the value in *dz* with a scale factor *s3* ∈ {1, ..., 8} and then adding an unsigned displacement *u4* ∈ {0, ..., 15} to the product. |
| Precodition | if *px* = *frame*, the processor must be in a regular state (*RC* or *RU*) |
| Faults | *privv* if the instruction is a privileged case<br>*drfnu* if *px* contains the null value<br>*wrptv* if *px* holds a read only pointer<br>*drfid* if *px* holds an id pointer<br>*sterr* if *px* = *frame* and in a quarantine state<br>*drfcd* if *px* refers to a code object<br>*sealv* if *px* refers to a sealed object<br>*ixoob* if the result of the index expression exceeds the size of a word (i.e. if carries occur)<br>*ixoob* if index ≥ π<br>*fram0* if *px* refers to a terminal or an ordinary frame and *py* = *rcd*<br>*fram0* if *px* refers to a gate frame and *py* ≠ *rcd* and index = 0 |

### 3.2.4 Attribute instructions (ATU)

| | |
|---|---|
| Instruction | **`qpi dy = px`** Query π attribute |
| Description | Query the π attribute of the object referred to by *px* and write the result to *dy*. |
| Precodition | if *px = frame*, the processor must be in a regular state (*RC* or *RU*) |
| Faults | *privv* if the instruction is a privileged case<br>*drfnu* if *px* contains the null value<br>*drfid* if *px* holds an id pointer<br>*sterr* if *px = frame* and in a quarantine state<br>*drfcd* if *px* refers to a code object |

| | |
|---|---|
| Instruction | **`qdtb dy = px`** Query δ attribute in number of bytes<br>**`qdth dy = px`** Query δ attribute in number of half words<br>**`qdtw dy = px`** Query δ attribute in number of words<br>**`qdtd dy = px`** Query δ attribute in number of double words |
| Description | Query the δ attribute of the object referred to by *px*, divide the value by 1 (*qdtb*), 2 (*qdth*), 4 (*qdtw*) or 8 (*qdtd*) and write the result to *dy*. The remainder of the division is discarded. |
| Precodition | if *px = frame*, the processor must be in a regular state (*RC* or *RU*) |
| Faults | *privv* if the instruction is a privileged case<br>*drfnu* if *px* contains the null value<br>*drfid* if *px* holds an id pointer<br>*sterr* if *px = frame* and in a quarantine state<br>*drfcd* if *px* refers to a code object |
| Remarks | The instruction *qdt dy = px* is provided as a pseudo-instruction and implemented as *qdtb dy = px*. |

| | |
|---|---|
| Instruction | **`nchk px`** Null check |
| Description | Check whether *px* = null |
| Faults | *drfnu* if *px* contains the null value |
| Remarks | Usually used for method inlining. The instruction is more efficient than a corresponding dummy load because it does not require the corresponding object's attributes. Furthermore, it also works with pointers to empty objects. |

### 3.2.5 Branch instructions (BPU)

| | |
|---|---|
| Instruction | **`beqp px,py,sd12`** Branch if pointers are equal<br>**`bnep px,py,sd12`** Branch if pointers are not equal<br>**`beq dy,dz,sd12`** Branch if equal<br>**`bne dy,dz,sd12`** Branch if not equal<br>**`bgeu dy,dz,sd12`** Branch if greater or equal unsigned<br>**`bges dy,dz,sd12`** Branch if greater or equal signed<br>**`bltu dy,dz,sd12`** Branch if less than unsigned<br>**`blts dy,dz,sd12`** Branch if less than signed |
| Description | Branch to the target code index within the current code object if the two register operands meet a given condition. The target code index is obtained by sign-extending and adding *sd12* to the current code index (code index of the conditional branch instruction). |
| Faults | *tciob* if the target code index is greater or equal the size of the current code object Ξ |

| | |
|---|---|
| Instruction | **`bra sd25`** Branch unconditionally |
| Description | Branch to the target code index within the current code object. The target code index is obtained by sign-extending and adding *sd25* to the current code index (code index of the *bra* instruction). |
| Faults | *tciob* if the target code index is greater or equal the size of the current code object Ξ |

| | |
|---|---|
| Instruction | **`jmp dy`** Jump unconditionally |
| Description | Jump to target code index *dy* within the current code object. |
| Faults | *tciob* if the target code index is greater or equal the size of the current code object Ξ |

| | |
|---|---|
| Instruction | **`bsr sd25`** Branch to subroutine |
| Description | Branch to a subroutine at a target code index within the current code object. The target code index is obtained by sign-extending and adding *sd25* to the current code index (code index of the *bsr* instruction). |
| Precodition | regular state *RC* or *RU* |
| Postcondition | quarantine state *QC* or *QU*, *rix* set to the code index of the instruction following the *bsr* instruction |
| Faults | *sterr* if executed in a quarantine state<br>*tciob* if the target code index is greater or equal the size of the current code object Ξ |
| Remarks | Parameters are passed via registers and, if required, a parameter object. The callee cannot access the caller's frame. |

| | |
|---|---|
| Instruction | `jsr dy` Jump to subroutine |
| Description | Jump to a subroutine at target code index *dy* within the current code object. |
| Precodition | regular state *RC* or *RU* |
| Postcondition | quarantine state *QC* or *QU*, *rix* set to the code index of the instruction following the *jsr* instruction |
| Faults | *sterr* if executed in a quarantine state<br>*tciob* if the target code index is greater or equal the size of the current code object Ξ |
| Remarks | Parameters are passed via registers and, if required, a parameter object. The callee cannot access the caller's frame. |

| | |
|---|---|
| Instruction | `rts` Return from subroutine |
| Description | Return from a subroutine to the caller within the current code object and set *rix* to *zero*. |
| Precodition | quarantine state *QC* or *QU*, *rix* ≠ *zero* |
| Postcondition | regular state *RC* or *RU*, *rix* = *zero* |
| Faults | *sterr* if executed in regular state *RC* or *RU* or in library entry quarantine *QL*<br>*rixeq* if *rix* is *zero* |
| Rationale | By setting *rix* to *zero*, *rts* "consumes" the *rix* value and ensures that the value is only used once and that *rts* returns to the actual caller. |
| Remarks | Attempting to leave a library entry routine with *rts* raises a *sterr* fault because *rts* is executed in state *QL* instead *QC* or *QU*.<br>If a routine does not save *rix* before it calls another subroutine (or itself), it will be impossible to leave the subroutine without raising a *rixeq* fault. |

| | |
|---|---|
| Instruction | `check rcd` Check *rcd* |
| Description | Check that *rcd* refers to the current code object. |
| Precodition | state *RU*, *rcd* = *code* |
| Postcondition | state *RC* |
| Faults | *sterr* if executed in *RC*, *QC*, *QU*, *QL*<br>*rcdnc* if *rcd* ≠ *code* |
| Remarks | Required before calling routines in a library *B* from a library *A* |

| | |
|---|---|
| Instruction | `jlib px,ix20` Jump to library entry routine<br>`jlib px,dy` Jump to library entry routine |
| Description | Branch to a library entry routine in the code object referred to by *px* at the target code index given as an unsigned index *ix20* or the contents of a data register *dy*. |
| Precodition | regular state *RC* (*rcd* = *code*), *px* refers to a code object, *px* ≠ *code*, *px* ≠ *null* |
| Postcondition | library entry quarantine *QL*, *rix* set to the code index of the instruction following the *jlib* instruction |
| Faults | *sterr* if executed in state *RU*, *QU*, *QC*, *QL*<br>*tcoil* if *px* does not refer to a code object or if *px* refers to the current code object or if *px* = *null*<br>*tciob* if the target code index is greater or equal the public size ξ of the target code object |
| Rationale | Actually, *jlib* should write both *rcd* and *rix* to save the return code object along with the return index. To provide for efficient implementations (i.e. ensure that each instruction writes at most one register), *jlib* merely writes *rix* and requires that the current code pointer has beforehand been saved to *rcd* with the *cpfc* instruction. In return, *jlib* must check that *rcd* actually refers to the current code object. For this task, *jlib* would require three pointer parameters (*frame* for state checking, *px*, and *rcd*). Again, to provide for efficient implementations (i.e. ensure that each instruction reads at most two pointer registers), the required check is implemented by a separate *check* instruction that verifies that *rcd* equals the current code pointer *code* and that triggers a transition from state regular unchecked *RU* to state regular checked *RC*.<br>Libraries are not allowed to call themselves with the *jlib* instruction. If they did, *rtlb* could not decide whether *rcd* refers to the actual caller or has not been properly restored. |
| Remarks | Parameters are passed via registers and, if required, a parameter object. The callee cannot access the caller's frame.<br>Application code is always in state regular *RC* or in state quarantine *QC*, and *rcd* always holds a pointer to the current code object. |

| | |
|---|---|
| Instruction | `rtlb` Return from library entry routine |
| Description | Return from a library entry routine to the caller and set *rix* to *zero*. |
| Precodition | quarantine state *QL*, *rix* ≠ *zero*, *rcd* ≠ *null*, *rcd* ≠ *code* |
| Postcondition | regular state *RC*, *rix* = *zero* |
| Faults | *sterr* if executed in state *RC*, *QC*, or *RU*, *QU*<br>*tcoil* if *rcd* = *code* or *rcd* = *null*<br>*rixeq* if *rix* is *zero* |
| Rationale | By setting *rix* to *zero*, *rtlb* "consumes" the *rix* value and ensures that the value is only used once and that *rtlb* returns to the actual caller. |
| Remarks | Attempting to leave a library entry routine with *rts* raises a *sterr* fault because *rts* is executed in state *QL* instead *QC* or *QU*.<br>If a routine does not save *rix* before it calls another subroutine (or itself), it will be impossible to leave the subroutine without raising a *rixeq* fault. |

| | |
|---|---|
| Instruction | `trap ui10` System call |
| Description | Call a system function in the *core object*. The parameter *ui10* may be used to differentiate classes of system calls. |

## 3.2.6 Privileged instructions (ALU)

| | |
|---|---|
| Instruction | `ctsr  xx = dz` Copy to system register<br>`cfsr  dx = xz` Copy from system register |
| Description | |
| Remarks | |

## 3.2.7 Privileged instructions (PGU)

| | |
|---|---|
| Instruction | `crop  py = px` Copy to system register<br>`cidp  py = px` Copy from system register |
| Description | |
| Remarks | |

| | |
|---|---|
| Instruction | `rpr   py = px` Restore pointer rights |
| Description | |
| Remarks | |

| | |
|---|---|
| Instruction | `seal  px` Seal object<br>`unsl  px` Unseal object |
| Description | |
| Remarks | |

| | |
|---|---|
| Instruction | `alcb  frame = dy` Allocate stack bumper |
| Description | |
| Remarks | |

| | |
|---|---|
| Instruction | `ciop  px = dy,dz` Create io Pointer |
| Description | |
| Remarks | |

| | |
|---|---|
| Instruction | `ccp   px = dy` Create code pointerr |
| Description | |
| Remarks | |

## 3.2.8 Privileged instructions (LSU)

| | |
|---|---|
| Instruction | `flshic dz` Flush instruction cache line<br>`flshdc dz` Flush data cache line<br>`flshac dz` Flush attribute cache line<br>`flshbc dz` Flush branch target cache line |
| Description | |
| Remarks | |

| | |
|---|---|
| Instruction | `pcce  dz = px` Put context cache entry |
| Description | |
| Remarks | |

| | |
|---|---|
| Instruction | `rcce  dz` Remove context cache entry |
| Description | |
| Remarks | |

| | |
|---|---|
| Instruction | `ccc` Clear context cache |
| Description | |
| Remarks | |

| | |
|---|---|
| Instruction | `ciop  px = dy,dz` Create io Pointer |
| Description | |
| Remarks | |

| | | |
|---|---|---|
| Instruction | `ccp px = dy` | Create code pointerr |
| Description | | |
| Remarks | | |

## 3.2.9 Privileged instructions (ATU)

| | | |
|---|---|---|
| Instruction | `qpir dy = px`<br>`qdtr dy = px` | Query raw π attribute<br>Query raw δ attribute |
| Description | | |
| Remarks | | |

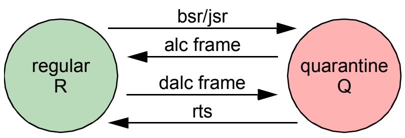
| | | |
|---|---|---|
| Instruction | `qptr dy = px` | Query raw pointer |
| Description | | |
| Remarks | | |

## 3.2.10Privileged instructions (BPU)

| | | |
|---|---|---|
| Instruction | `sync` | Sync |
| Description | | |
| Remarks | | |

| | | |
|---|---|---|
| Instruction | `rte` | Return from Exception |
| Description | | |
| Remarks | | |

## Stack protection mechanism: Application code



## State privileges and state transitions

| | state | bsr/jsr | rts | alc frame | dalc frame | deref frame |
|---|---|---|---|---|---|---|
| R | regular | →Q | × | × | →Q | √ |
| Q | quarantine | × | →R | →R | × | × |

Terminal subroutine that requires no stack space

```
subrt:    ...                        a subroutine that does not require stack space can remain in quarantine
          rts                        rts returns to regular state
```

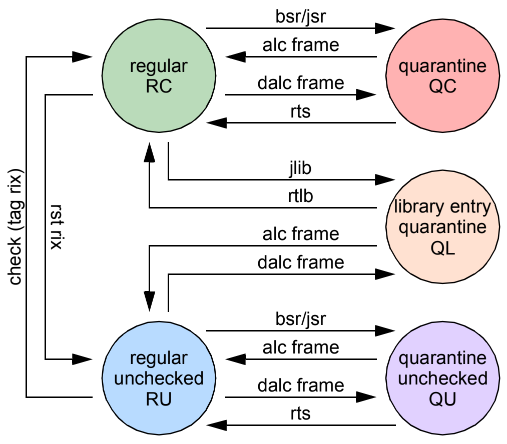
Terminal subroutine that requires stack space (does not call subroutines or routines in libraries)

```
subrt:    alct   frame = π,δ         terminal frame without a slot for rix (impossible to save rix)
          ...
          ...                        (should a subroutine be called, rix is cleared and rts will raise a rixeq fault)
          dalc   frame
          rts
```

Standard subroutine (may call subroutines and routines in other libraries)

```
subrt:    alc    frame = π,δ         regular frame with reserved slot for rix
          sw     frame[0] = rix      save rix
          ...
          bsr    ...
          ...
          jlib   ...
          ...
          lw     rix = frame[0]      restore rix
          dalc   frame
          rts
```

## Stack protection mechanism: Library code



## State privileges and state transitions

| | state | bsr/jsr | jlib | rts | rtlb | alc frame | dalc frame | check rcd | rst rix | rst rcd | deref frame |
|---|---|---|---|---|---|---|---|---|---|---|---|
| RC | regular checked | →QC | →QL | × | × | × | →QC | × | →RC/RU | × | √ |
| RU | regular unchecked | →QU | × | × | × | × | →QL/QU | →RC | √ | √ | √ |
| QC | quarantine checked | × | × | →RC | × | →RC | × | × | × | × | × |
| QU | quarantine unchecked | × | × | →RU | × | →RU | × | × | × | × | × |
| QL | library entry quarantine | × | × | × | →RC | →RU | × | × | × | × | × |

Terminal library routine that requires no stack space

```
libentry: ...                      libinner: ...
          rtlb                               rts
```

Terminal library routine that requires stack space

```
libentry: alct  frame = π,δ        libinner: alct  frame = π,δ      terminal frame without a slot for rix
          ...                                ...
          dalc  frame                        dalc  frame
          rtlb                               rts
```

Unchecked library routine (may call subroutines, but no routines in other libraries)

```
libentry: alc   frame = π,δ        libinner: alc   frame = π,δ      ordinary frame with reserved slot for rix
          sv    rix                          sv    rix              save rix
          ...                                ...
          bsr   other                        bsr   other
          ...                                ...
          rst   rix                          rst   rix              restore rix
          dalc  frame                        dalc  frame
          rtlb                               rts
```

Gate library routine to enter state checked (may call subroutines and routines in other libraries)

```
libentry: alcg  frame = π,δ        libinner: alcg  frame = π,δ      gate frame with reserved slots for rix, rcd
          sv    rix                          sv    rix              save rix
          sv    rcd                          sv    rcd              save rcd
          cpfc  rcd                          cpfc  rcd              copy code to rcd
          check rcd                          check rcd              enter checked state
          ...                                ...
          jlib  px,other                     jlib  px,other
          ...                                ...
          bsr   other                        bsr   other
          ...                                ...
          rst   rix                          rst   rix              restore rix and unchecked state
          rst   rcd                          rst   rcd              restore rcd
          dalc  frame                        dalc  frame
          rtlb                               rts
```

## Invariants

- *rix* contains the return index from the caller or is *zero*
  - *rix* = *zero* after calling a subroutine or a library entry routine
  - *rix* = *zero* if *rix* is not saved to but restored from *frame*[0]
- *rcd* refers to the calling code object, to the current code object or is *null*
  - *rcd* = *code* in the checked states *RC*, *QC* (and therefore in application code objects)
  - *rcd* = *null* if *rcd* is not saved to but restored from *frame*[0]

## Assembler-Syntax: Labels

| kind | | definition | reference | value |
|---|---|---|---|---|
| physical | privileged code object (supervisor) | @physical> | @physical | physical byte address |
| | unprivileged code object (user) | @physical: | | |
| global | | global: | @physical.global<br>global | word index (into code object) |
| local | | .local: | @physical.global.local<br>global.local<br>.local | word index (into code object) |
| index | | ->index: | @physical.global->index<br>global->index<br>@physical.global.local->index<br>global.local->index<br>.local->index | pointer: pointer index (into object)<br>sword, uword: word index (into object)<br>shalf, uhalf: half word index (into object)<br>sbyte, ubyte: byte index (into object) |
| object attributes<br>(*x* = pi, dt, dtb, dth, dtw, dtl) | | *implicit* | @physical.global’*x*<br>global’*x*<br>@physical.global.local’*x*<br>global.local’*x*<br>.local’*x* | pi: number of pointers<br>dt: number of bytes<br>dtb: number of bytes<br>dth: number of half words<br>dtw: number of words<br>dtl: number of long words |
| code object attributes<br>(*y* = xi, cxi) | | *implicit* | @physical’*y* | xi: number of instructions (words)<br>cxi: number of instructions (words) |

## Fault table

| no | 5code | u/s | name | instructions | remark |
|---|---|---|---|---|---|
| 01 | panic | u/s | invariant violation | | should never happen |
| 02 | sverr | s | supervisor error | some privileged instructions | faults specific to privileged instructions |
| 03 | sterr | u/s | state error | any state dependent instruction | |
| 04 | illeg | u/s | illegal instruction | illegal | |
| 05 | privv | u | privilege violation | any privileged instruction | |
| 06 | tcoil | u/s | target code object illegal | jlib | no pointer to code object, equals code, equals null |
| 07 | tciob | u/s | target code index out of bounds | bra, bcc, jmp, bsr, jsr, jlib | target index $\geq \Xi$(code); *jlib*: target index $\geq \xi$(code) |
| 08 | endoc | u/s | end of code | any | control flow reaching end of code object |
| 09 | rixeq | u/s | rix equal zero | rts, rtlb | |
| 0a | rcdnu | u/s | rcd null | rtlb | |
| 0b | rcdnc | u/s | rcd not code | check | |
| 0c | drfnu | u/s | deref null | load, store, qxx | attempt to dereference a null pointer |
| 0d | drfid | u | deref id (privv) | load, store, qxx | attempt to dereference an id pointer |
| 0e | drfcd | u | deref code (privv) | load, store, qxx | attempt to dereference a code pointer |
| 0f | wrptv | u | write protection violation (privv) | store | attempt to write to a read only object |
| 10 | sealv | u | seal violation (privv) | store | attempt to write to a sealed object |
| 11 | ixoob | u/s | index out of bounds | load, store | index $\geq \pi$, $\delta$ |
| 12 | frtyp | u/s | frame type | sv, rst | *sv rix* or *rst rix* in a terminal frame<br>*sv rcd* or *rst rcd* in an ordinary or terminal frame<br>never state dependent (*sterr* has a higher priority) |
| 13 | aperr | u/s | alc paramter error | alc | general: $\pi \geq 2^{29}$, $\delta \geq 2^{31}$ |
| 14 | hpovf | u/s | heap overflow | alc | |
| 15 | stovf | u/s | stack overflow | alc frame | |
| 16 | ccmis | u/s | context cache miss | gcp | |
| 17 | break | u/s | breakpoint | any | |

## Causes for ***sverr*** (ToDo)

*todo*

## Causes for ***panic*** (ToDo)

*panic* if the pointer to be copied is an illegal null pointer
*panic* if the pointer refers to an uninitialized object in user mode
*panic* if the pointer refers to a stack frame object

*alc frame*, *alct frame*, *alcg frame*, *dalc frame: panic* if *frame* = null or if *frame* does not hold a frame pointer

## Instruction Formats

| F | primary7 |  | x5 | secondary7 |  | z5 | y5 | func | For primary |
|---|---|---|---|---|---|---|---|---|---|
|  | 31:29 | 28:25 | 24:20 | 19:16 | 15:13 | 12:8 | 7:3 | 2:0 |  |
| A | 0 | f | x | h | 0 | z/ud5 | y | g | reg 0 |
| B | 0 | f | x | ud4 | s3 | z | y | g | lsu A, lsu B |
| C | 0 | f | x | ui12, ui12/si12, ix12, sd12 |  |  | y | g | alu 1, alu 2, lsu 8, lsu 9, bpu C (0,1) |
| D | 0 | f | sd12$_L$ | sd12$_H$ |  | z | y | g | bpu C (2–7) |
| E | 0 | f | x | δ15$_H$ |  |  | y | δ15$_L$ | pgu 4 (x\f) |
| F | 0 | f | x | π15$_H$ |  | z | π15$_L$ |  | pgu 5 (x\f) |
| G | 0 | f | x | ui20, π9:δ11, ix20 |  |  |  |  | alu 3, pgu 4 (x=f), pgu 5 (x=f), pgu 6, bpu F |
| H | 0 | f | sd25 |  |  |  |  |  | bpu D, bpu E |

## Embedded immediates, indexes, displacements

| | | |
|---|---|---|
| ui5 | 5-bit unsigned immediate | zero-extended |
| ui12 | 12-bit unsigned immediate | zero-extended |
| ui20 | 20-bit upper immediate | shifted left by 12 positions |
| si12 | 12-bit signed immediate | sign-extended |
| ix12 | 12-bit unsigned object index | zero-extended, implicitly scaled (unless privileged access via *null*) |
| ix20 | 20-bit unsigned code object index | zero-extended, implicitly scaled by 2 |
| ud4 | 4-bit unsigned displacement | zero-extended, implicitly scaled (unless privileged access via *null*) |
| s3 | 3-bit unsigned scale factor | scale factor 1...8 (8 encoded as 000), implicitly scaled (unless privileged access via *null*) |
| sd12 | 12-bit signed branch displacement | sign-extended, implicitly scaled by 2 |
| sd25 | 25-bit signed branch displacement | sign-extended, implicitly scaled by 2 |
| δ15 | 15-bit unsigned delta | zero-extended |
| π15 | 15-bit unsigned pi | zero-extended |
| π9:δ11 | 9 bit unsigned π (bits 19:11), 11 bit unsigned δ (bits 10:0) | both zero-extended |
| δ15$_H$ | 15-bit unsigned δ, higher 12 bits 14:3 | |
| δ15$_L$ | 15-bit unsigned δ, lower 3 bits 2:0 | |
| π15$_H$ | 15-bit unsigned π, higher 7 bits 14:8 | |
| π15$_L$ | 15-bit unsigned π, lower 8 bits 7:0 | |
| sd12$_H$ | 12-bit signed displacement, higher 7 bits 11:5 | |
| sd12$_L$ | 12-bit signed displacement, lower 5 bits 4:0 | |

## Opcode map (32 Bit)

| | f7 | | x5 | h7 | | z5 | y5 | g3 | Instruction | A | B | P | α | Q | W | EX | ME | AC |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 31:29 | 28:25 | 24:20 | 19:17 | 16:13 | 12:8 | 7:3 | 2:0 | | | | | | | | | | |
| n | = | reg 0 | x\r | = | alu 00 | z\r | y\r | 0 | **`add    dx = dy,dz`** | dy | dz | – | – | – | dx | D | – | – |
| | | | | | | | | 1 | **`sub    dx = dy,dz`** | | | | | | | | | |
| | | | | | | | | 4 | **`and    dx = dy,dz`** | | | | | | | | | |
| | | | | | | | | 5 | **`andn   dx = dy,dz`** | | | | | | | | | |
| | | | | | | | | 6 | **`or     dx = dy,dz`** | | | | | | | | | |
| | | | | | | | | 7 | **`xor    dx = dy,dz`** | | | | | | | | | |
| | | | | | alu 01 | | | 0 | **`sltu   dx = dy,dz`** | dy | dz | | | | | | | |
| | | | | | | | | 1 | **`slts   dx = dy,dz`** | | | | | | | | | |
| | | | | | alu 02 | | | 0 | **`mul    dx = dy,dz`** | dy | dz | | | | | | | |
| | | | | | | | | 1 | **`mulhs  dx = dy,dz`** | | | | | | | | | |
| | | | | | | | | 2 | **`mulhu  dx = dy,dz`** | | | | | | | | | |
| | | | | | | | | 3 | **`mulhsu dx = dy,dz`** | | | | | | | | | |
| | | | | | alu 03 | | | | | dy | dz | | | | | | | |
| | | | | | alu 04 | | | 0 | **`srl    dx = dy,dz`** | dy | dz | | | | | | | |
| | | | | | | | | 1 | **`sra    dx = dy,dz`** | | | | | | | | | |
| | | | | | | | | 2 | **`sll    dx = dy,dz`** | | | | | | | | | |
| | | | | | | | | 3 | ***`rol    dx = dy,dz`*** | | | | | | | | | |
| | | | | | alu 05 | ui5 | | 0 | **`srli   dx = dy,ui5`** | dy | # | | | | | | | |
| | | | | | | | | 1 | **`srai   dx = dy,ui5`** | | | | | | | | | |
| | | | | | | | | 2 | **`slli   dx = dy,ui5`** | | | | | | | | | |
| | | | | | | | | 3 | ***`roli   dx = dy,ui5`*** | | | | | | | | | |
| | | | | | alu 06 | | | | | dy | # | | | | | | | |
| | | | | | alu 07 | z\r | = | 0 | **`not    dx = dz`** | – | dz | | | | | | | |
| | | | | | | | | 1 | **`extbs  dx = dz`** | | | | | | | | | |
| | | | | | | | | 2 | **`exthu  dx = dz`** | | | | | | | | | |
| | | | | | | | | 3 | **`exths  dx = dz`** | | | | | | | | | |
| | | | | | | | | 5 | **`clz    dx = dz`** | | | | | | | | | |
| p | = | reg 0 | x | = | sys 08 | z\r | = | 0 | `ctsr   xx = dz` | – | dz | – | – | – | sx | D | – | – |
| | | | x\r | | | z | | 1 | `cfsr   dx = xz` | | sz | – | | | dx | D | | |
| | | | x=0! | | | z\r | | 4 | `clric  dz (deprecated)` | | dz | px | | | – | – | | |
| | | | x=0! | | | z\r | | 5 | `clrbc  dz (deprecated)` | | dz | px | | | – | – | | |
| | | | x=0! | | | z\r | | 6 | `flshdc dz (deprecated)` | | dz | px | | | – | – | | |
| | | | x=0! | | | z\r | | 7 | `flshac dz (deprecated)` | | dz | px | | | – | – | | |
| p | = | reg 0 | x\f,r | = | sys 09 | z\r | = | 0 | `pcce   dz,px` | – | dz | px | – | – | – | – | – | – |
| | | | = | | | z\r | | 1 | `rcce   dz` | | dz | – | | | | | | |
| | | | = | | | = | | 2 | `clrcc` | | – | – | | | | | | |
| | | | = | | | = | | 4 | `clric` | | – | – | | | | | | |
| | | | = | | | = | | 5 | `clrbc` | | – | – | | | | | | |
| | | | = | | | = | | 6 | `flshdc` | | – | – | | | | | | |
| | | | = | | | = | | 7 | `flshac` | | – | – | | | | | | |
| af | = | reg 0 | x\0$^5$ | = | atu 0A | = | y\r | 0 | **`qpi    dy = px`** | – | ***dr***$^7$ | px | αx | – | dy | D | – | – |
| p | | | x$^5$ | | | | | 1 | `qpir   dy = px` | | – | | αx | | | | | |
| p | | | x$^5$ | | | | | 2 | `qdtr   dy = px` | | – | | αx | | | | | |
| p | | | x$^5$ | | | | | 3 | `qptr   dy = px` | | – | | – | | | | | |
| af | | | x\0$^5$ | | | | | 4 | **`qdtb   dy = px`** | | ***dr***$^7$ | | αx | | | | | |
| af | | | x\0$^5$ | | | | | 5 | **`qdth   dy = px`** | | ***dr***$^7$ | | αx | | | | | |
| af | | | x\0$^5$ | | | | | 6 | **`qdtw   dy = px`** | | ***dr***$^7$ | | αx | | | | | |
| af | | | x\0$^5$ | | | | | 7 | **`qdtd   dy = px`** | | ***dr***$^7$ | | αx | | | | | |
| n | = | reg 0 | x\f,r | = | atu 0B | = | = | 0 | **`nchk   px`** | – | – | px | – | – | – | – | – | – |
| n/p | = | reg 0 | x\f$^6$ | = | pgu 0C | = | y\f$^6$ | 0 | **`cpp    py = px`** | – | – | px | αx | – | py | Pα | – | – |
| p | | | x\0,f,r | | | | y\f,r | 1 | `crop   py = px` | | | | | | py | Pα | | |
| p | | | x\0,f,r | | | | y\f,r | 2 | `cidp   py = px` | | | | | | py | Pα | | |
| p | | | x\0,f,r | | | | y\f,r | 3 | `rpr    py = px` | | | | | | py | Pα | | |
| p | | | x\0,f,r | | | | = | 4 | `seal   px` | | | | | | αx | α! | | |
| p | | | x\0,f,r | | | | = | 5 | `unsl   px` | | | | | | αx | α! | | |
| n | = | reg 0 | x\f,r | = | pgu 0D | z\r | y\r | 0 | **`alc    px = dy,dz`** | dy | dz | ar | – | – | px | Pα | – | – |
| p | | | x\f,r | | | z\r | y\r | 1 | `ciop   px = dy,dz` | dy | dz | – | – | | px | Pα | | – |
| p | | | x\f | | | = | y\r | 2 | `ccp    px = dy` | dy | – | – | – | | px | P | | α |
| p | | | x=f! | | | = | y\r | 3 | `alcb   px = dy` | dy | – | – | – | | px | P | | α |
| n | | | x\f,r | | | = | = | 4 | **`gcp    px`** | – | – | – | – | | px | P | | α |
| m | | | x=f! | | | = | = | 5 | **`dalc   frame`** | – | **dr** | **px** | αx | | px | P | | α |
| n | | | x=r! | | | = | = | 6 | **`cpfc   rcd`** | – | – | – | – | | px | Pα | | – |
| m | = | reg 0 | x=r! | = | bpu 0E | = | = | 0 | **`check  rcd`** | – | **dr** | px | – | **pf** | **dr** | D | – | – |
| n | | | = | | | ui10 | ui10 | 4 | **`trap   ui10`** | – | – | – | – | – | – | – | | |
| p | | | = | | | = | = | 6 | `sync` | # | – | – | – | – | – | – | | |
| p | | | = | | | = | = | 7 | `rte` | eci | es | ecd | αc | – | st | D | | |
| n | = | reg 0 | = | = | bpu 0F | = | y\r | 0 | **`jmp    dy`** | dy | – | – | – | – | – | – | – | – |
| m | | | = | | | | y\r | 4 | **`jsr    dy`** | dy | **dr** | – | – | **pf** | **dr** | D | | |
| m | | | = | | | | y=r! | 5 | **`rts`** | dy | **dr** | – | – | **pf** | **dr** | D | | |
| m | | | x\0,f,r | | | | y\r | 6 | **`jlib   px,dy`** | dy | **dr** | px | αx | **pf** | **dr** | D | | |
| m | | | x=r! | | | | y=r! | 7 | **`rtlb`** | dy | **dr** | px | αx | **pf** | **dr** | D | | |

| | f7 | | x5 | h7 | | z5 | y5 | g3 | Instruction | A | B | P | α | Q | W | EX | ME | AC |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 31:29 | 28:25 | 24:20 | 19:17 | 16:13 | 12:8 | 7:3 | 2:0 | | | | | | | | | | |
| n | = | alu 1 | x\r | ui12 | | | y\r | 0<br>1<br>4<br>5<br>6<br>7 | **addi dx = dy,ui12**<br>**subi dx = dy,ui12**<br>**andi dx = dy,ui12**<br>**andni dx = dy,ui12**<br>**ori dx = dy,ui12**<br>**xori dx = dy,ui12** | dy | # | – | – | – | dx | D | – | – |
| | | alu 2 | | ui12/si12 | | | | 0<br>1<br>4<br>5 | **sltiu dx = dy,ui12**<br>**sltis dx = dy,si12**<br>**muliu dx = dy,ui12**<br>**mulis dx = dy,si12** | dy | # | – | – | – | dx | D | – | – |
| | | alu 3 | | ui20 | | | | | **lui dx = ui20** | – | # | – | – | – | dx | D | – | – |
| n | = | pgu 4 | x\f,r | $\delta 15_H$ | | | y\r | $\delta 15_L$ | **alc px = dy,δ15** | dy | # | ar | | | px | Pα | – | – |
| m | | | x=f! | π9 : δ11 | | | | | **alct frame = π9,δ11** | # | **dr** | **px** | αx | – | **px** | | | |
| n | = | pgu 5 | x\f,r | $\pi 15_H$ | | z\r | $\pi 15_L$ | | **alc px = π15,dz** | # | dz | ar | – | – | px | | | |
| m | | | x=f! | π9 : δ11 | | | | | **alcg frame = π9,δ11** | # | **dr** | **px** | αx | – | **px** | | | |
| n | | pgu 6 | x\f,r | | | | | | **alc px = π9,δ11** | # | – | ar | – | – | px | | | |
| m | | | x=f! | | | | | | **alc frame = π9,δ11** | # | **dr** | **px** | αx | – | **px** | | | |
| af<br>/p | = | lsu 8 | $x^{45}$ | ix12 | | | y\r<br>y\r<br>$y^{1?}$ | 0,1<br>2,3<br>5 | **lbu/s dy = px[ix12]**<br>**lhu/s dy = px[ix12]**<br>**lw dy = px[ix12]** | – | ***dr***$^7$ | px | αx | – | dy | – | D | – |
| af<br>/p | | lsu 9 | $x^{45}$ | ix12 | | | y\r<br>y\r<br>$y^{1?}$ | 0<br>1<br>2 | **sb px[ix12] = dy**<br>**sh px[ix12] = dy**<br>**sw px[ix12] = dy** | dy | ***dr***$^7$ | px | αx | – | – | – | – | – |
| af<br>/p | | lsu 9 | $x\backslash 0^5$ | ix12 | | | $y^{13}$ | 4<br>5<br>6<br>7 | **lp py = px[ix12]**<br>**lcp py = px[ix12]**<br>**sp px[ix12] = py**<br>**scp px[ix12] = py** | – | ***dr***$^7$ | px | αx | –<br>null<br>py<br>py | py<br>py<br>–<br>py | – | P<br>P<br>–<br>null | α<br>α<br>–<br>(α) |
| n<br>/p | | lsu A | $x\backslash f^{45}$ | s3 | ud4 | z\r | y\r<br>y\r<br>$y^{2?}$ | 0,1<br>2,3<br>5 | **lbu/s dy = px[dz*s3+ud4]**<br>**lhu/s dy = px[dz*s3+ud4]**<br>**lw dy = px[dz*s3+ud4]** | – | dz | px | αx | – | dy | – | D | – |
| n<br>/p | | lsu B | $x\backslash f^{45}$ | | | | y\r<br>y\r<br>$y^{2?}$ | 0<br>1<br>2 | **sb px[dz*s3+ud4] = dy**<br>**sh px[dz*s3+ud4] = dy**<br>**sw px[dz*s3+ud4] = dy** | dy | dz | px | αx | – | – | – | – | – |
| n<br>/p | | lsu B | $x\backslash 0,f^5$ | | | | $y\backslash f^2$ | 4<br>5<br>6<br>7 | **lp py = px[dz*s3+ud4]**<br>**lcp py = px[dz*s3+ud4]**<br>**sp px[dz*s3+ud4] = py**<br>**scp px[dz*s3+ud4] = py** | – | dz | px | αx | –<br>null<br>py<br>py | py<br>py<br>–<br>py | – | P<br>P<br>–<br>null | α<br>α<br>–<br>(α) |
| a | = | lsu A | x=f! | = | = | z=0! | y=r! | 5 | **rst rix** | – | dz | px | αx | – | dy | – | D | – |
| | | lsu B | | | | | | 2 | **sv rix** | dy | dz | px | αx | – | – | – | – | – |
| | | lsu B | | | | | | 4<br>6 | **rst rcd**<br>**sv rcd** | – | dz | px | αx | –<br>py | py<br>– | – | P<br>– | α<br>– |
| n | = | bpu C | x\f,r | sd12 | | | y\f,r | 0<br>1 | **beqp px,py,sd12**<br>**bnep px,py,sd12** | – | – | px | – | py | – | – | – | – |
| | | | $sd12_L$ | $sd12_H$ | | z\r | y\r | 2<br>3<br>4<br>5<br>6<br>7 | **beq dy,dz,sd12**<br>**bne dy,dz,sd12**<br>**bgeu dy,dz,sd12**<br>**bges dy,dz,sd12**<br>**bltu dy,dz,sd12**<br>**blts dy,dz,sd12** | dx | dy | – | – | – | – | – | – | – |
| n | | bpu D | sd25 | | | | | | **bra sd25** | – | – | – | – | – | – | – | – | – |
| m | | bpu E | | | | | | | **bsr sd25** | – | **dr** | – | – | **pf** | **dr** | D | – | – |
| m | | bpu F | $x\backslash 0,f^8$ | ix20 | | | | | **jlib px,ix20** | # | **dr** | px | αx | **pf** | **dr** | D | – | – |

## Opcode map (incl. DIV- and 64-Bit-Extension)

| | f7 | | x5 | h7 | | z5 | y5 | g3 | Instruction | A | B | P | α | Q | W | EX | ME | AC |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 31:29 | 28:25 | 24:20 | 19:17 | 16:13 | 12:8 | 7:3 | 2:0 | | | | | | | | | | |
| n | = | reg 0 | x\r | = | alu 00 | z\r | y\r | 0 | **`add dx = dy,dz`** | dy | dz | – | – | – | dx | D | – | – |
| | | | | | | | | 1 | **`sub dx = dy,dz`** | | | | | | | | | |
| | | | | | | | | 2 | `addd dx = dy,dz` | | | | | | | | | |
| | | | | | | | | 3 | `subd dx = dy,dz` | | | | | | | | | |
| | | | | | | | | 4 | **`and dx = dy,dz`** | | | | | | | | | |
| | | | | | | | | 5 | **`andn dx = dy,dz`** | | | | | | | | | |
| | | | | | | | | 6 | **`or dx = dy,dz`** | | | | | | | | | |
| | | | | | | | | 7 | **`xor dx = dy,dz`** | | | | | | | | | |
| | | | | | alu 01 | | | 0 | **`sltu dx = dy,dz`** | dy | dz | | | | | | | |
| | | | | | | | | 1 | **`slts dx = dy,dz`** | | | | | | | | | |
| | | | | | alu 02 | | | 0 | **`mul dx = dy,dz`** | dy | dz | | | | | | | |
| | | | | | | | | 1 | **`mulhs dx = dy,dz`** | | | | | | | | | |
| | | | | | | | | 2 | **`mulhu dx = dy,dz`** | | | | | | | | | |
| | | | | | | | | 3 | **`mulhsu dx = dy,dz`** | | | | | | | | | |
| | | | | | | | | 4 | `divu dx = dy,dz` | | | | | | | | | |
| | | | | | | | | 5 | `divs dx = dy,dz` | | | | | | | | | |
| | | | | | | | | 6 | `remu dx = dy,dz` | | | | | | | | | |
| | | | | | | | | 7 | `rems dx = dy,dz` | | | | | | | | | |
| | | | | | alu 03 | | | 0 | `muld dx = dy,dz` | dy | dz | | | | | | | |
| | | | | | | | | 1 | `mulhsd dx = dy,dz` | | | | | | | | | |
| | | | | | | | | 2 | `mulhud dx = dy,dz` | | | | | | | | | |
| | | | | | | | | 3 | `mulhsud dx = dy,dz` | | | | | | | | | |
| | | | | | | | | 4 | `divud dx = dy,dz` | | | | | | | | | |
| | | | | | | | | 5 | `divsd dx = dy,dz` | | | | | | | | | |
| | | | | | | | | 6 | `remud dx = dy,dz` | | | | | | | | | |
| | | | | | | | | 7 | `remsd dx = dy,dz` | | | | | | | | | |
| | | | | | alu 04 | | | 0 | **`srl dx = dy,dz`** | dy | dz | | | | | | | |
| | | | | | | | | 1 | **`sra dx = dy,dz`** | | | | | | | | | |
| | | | | | | | | 2 | **`sll dx = dy,dz`** | | | | | | | | | |
| | | | | | | | | 3 | ***`rol dx = dy,dz`*** | | | | | | | | | |
| | | | | | | | | 4 | `srld dx = dy,dz` | | | | | | | | | |
| | | | | | | | | 5 | `srad dx = dy,dz` | | | | | | | | | |
| | | | | | | | | 6 | `slld dx = dy,dz` | | | | | | | | | |
| | | | | | | | | 7 | *`rold dx = dy,dz`* | | | | | | | | | |
| | | | | | alu 05 | ui5 | | 0 | **`srli dx = dy,ui5`** | dy | # | | | | | | | |
| | | | | | | | | 1 | **`srai dx = dy,ui5`** | | | | | | | | | |
| | | | | | | | | 2 | **`slli dx = dy,ui5`** | | | | | | | | | |
| | | | | | | | | 3 | ***`roli dx = dy,ui5`*** | | | | | | | | | |
| | | | | | alu 06 | ui6 | | 0,1 | `srlid dx = dy,ui6` | dy | # | | | | | | | |
| | | | | | | | | 2,3 | `sraid dx = dy,ui6` | | | | | | | | | |
| | | | | | | | | 4,5 | `sllid dx = dy,ui6` | | | | | | | | | |
| | | | | | | | | 6,7 | *`rolid dx = dy,ui6`* | | | | | | | | | |
| | | | | | alu 07 | z\r | = | 0 | **`not dx = dz`** | – | dz | | | | | | | |
| | | | | | | | | 1 | **`extbs dx = dz`** | | | | | | | | | |
| | | | | | | | | 2 | **`exthu dx = dz`** | | | | | | | | | |
| | | | | | | | | 3 | **`exths dx = dz`** | | | | | | | | | |
| | | | | | | | | 4 | `extwu dy = dz` | | | | | | | | | |
| | | | | | | | | 5 | **`clz dx = dz`** | | | | | | | | | |
| | | | | | | | | 6 | `clzd dx = dz` | | | | | | | | | |
| p | = | reg 0 | x | = | sys 08 | z\r | = | 0 | `ctsr xx = dz` | – | dz | – | – | – | sx | D | – | – |
| | | | x\r | | | z | | 1 | `cfsr dx = xz` | | sz | – | | | dx | D | | |
| | | | x=0! | | | z\r | | 4 | `clric dz (deprecated)` | | dz | px | | | – | – | | |
| | | | x=0! | | | z\r | | 5 | `clrbc dz (deprecated)` | | dz | px | | | – | – | | |
| | | | x=0! | | | z\r | | 6 | `flshdc dz (deprecated)` | | dz | px | | | – | – | | |
| | | | x=0! | | | z\r | | 7 | `flshac dz (deprecated)` | | dz | px | | | – | – | | |
| p | = | reg 0 | x\f,r | = | sys 09 | z\r | = | 0 | `pcce dz,px` | – | dz | px | – | – | – | – | – | – |
| | | | = | | | z\r | | 1 | `rcce dz` | | dz | – | | | | | | |
| | | | = | | | = | | 2 | `clrcc` | | – | – | | | | | | |
| | | | = | | | = | | 4 | `clric` | | – | – | | | | | | |
| | | | = | | | = | | 5 | `clrbc` | | – | – | | | | | | |
| | | | = | | | = | | 6 | `flshdc` | | – | – | | | | | | |
| | | | = | | | = | | 7 | `flshac` | | – | – | | | | | | |
| af | = | reg 0 | x\0$^{5}$ | = | atu 0A | = | y\r | 0 | **`qpi dy = px`** | – | ***dr***$^{7}$ | px | αx | – | dy | D | – | – |
| p | | | x$^{5}$ | | | | | 1 | `qpir dy = px` | | – | | αx | | | | | |
| p | | | x$^{5}$ | | | | | 2 | `qdtr dy = px` | | – | | αx | | | | | |
| p | | | x$^{5}$ | | | | | 3 | `qptr dy = px` | | – | | – | | | | | |
| af | | | x\0$^{5}$ | | | | | 4 | **`qdtb dy = px`** | | ***dr***$^{7}$ | | αx | | | | | |
| af | | | x\0$^{5}$ | | | | | 5 | **`qdth dy = px`** | | ***dr***$^{7}$ | | αx | | | | | |
| af | | | x\0$^{5}$ | | | | | 6 | **`qdtw dy = px`** | | ***dr***$^{7}$ | | αx | | | | | |
| af | | | x\0$^{5}$ | | | | | 7 | **`qdtd dy = px`** | | ***dr***$^{7}$ | | αx | | | | | |
| n | = | reg 0 | x\f,r | = | atu 0B | = | = | 0 | **`nchk px`** | – | – | px | – | – | – | – | – | – |
| n/p | = | reg 0 | x\f$^{6}$ | = | pgu 0C | = | y\f$^{6}$ | 0 | **`cpp py = px`** | – | – | px | αx | – | py | Pα | – | – |
| p | | | x\0,f,r | | | | y\f,r | 1 | `crop py = px` | | | | | | py | Pα | | |
| p | | | x\0,f,r | | | | y\f,r | 2 | `cidp py = px` | | | | | | py | Pα | | |
| p | | | x\0,f,r | | | | y\f,r | 3 | `rpr py = px` | | | | | | py | Pα | | |
| p | | | x\0,f,r | | | | = | 4 | `seal px` | | | | | | αx | α! | | |
| p | | | x\0,f,r | | | | = | 5 | `unsl px` | | | | | | αx | α! | | |

| | f7 | | x5 | h7 | | z5 | y5 | g3 | Instruction | A | B | P | α | Q | W | EX | ME | AC |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 31:29 | 28:25 | 24:20 | 19:17 | 16:13 | 12:8 | 7:3 | 2:0 | | | | | | | | | | |
| n | = | reg 0 | x\f,r | = | pgu 0D | z\r | y\r | 0 | `alc    px = dy,dz` | dy | dz | ar | – | – | px | Pα | – | – |
| p | | | x\f,r | | | z\r | y\r | 1 | `ciop   px = dy,dz` | dy | dz | – | – | | px | Pα | | – |
| p | | | x\f | | | = | y\r | 2 | `ccp    px = dy` | dy | – | – | – | | px | P | | α |
| p | | | x=f! | | | = | y\r | 3 | `alcb   px = dy` | dy | – | – | – | | px | P | | α |
| n | | | x\f,r | | | = | = | 4 | `gcp    px` | – | – | – | – | | px | P | | α |
| m | | | x=f! | | | = | = | 5 | `dalc   frame` | – | dr | px | αx | | px | P | | α |
| n | | | x=r! | | | = | = | 6 | `cpfc   rcd` | – | – | – | – | | px | Pα | | – |
| m | = | reg 0 | x=r! | = | bpu 0E | = | = | 0 | `check  rcd` | – | dr | px | – | pf | dr | D | – | – |
| n | | | = | | | ui10 | ui10 | 4 | `trap   ui10` | – | – | – | – | – | – | – | | |
| p | | | = | | | = | = | 6 | `sync` | # | – | – | – | – | – | – | | |
| p | | | = | | | = | = | 7 | `rte` | eci | es | ecd | αc | – | st | D | | |
| n | = | reg 0 | = | = | bpu 0F | = | y\r | 0 | `jmp    dy` | dy | – | – | – | – | – | – | – | – |
| m | | | = | | | | y\r | 4 | `jsr    dy` | dy | dr | – | – | pf | dr | D | | |
| m | | | = | | | | y=r! | 5 | `rts` | dy | dr | – | – | pf | dr | D | | |
| m | | | x\0,f,r | | | | y\r | 6 | `jlib   px,dy` | dy | dr | px | αx | pf | dr | D | | |
| m | | | x=r! | | | | y=r! | 7 | `rtlb` | dy | dr | px | αx | pf | dr | D | | |
| n | = | alu 1 | x\r | ui12 | | | y\r | 0 | `addi   dx = dy,ui12` | dy | # | – | – | – | dx | D | – | – |
| | | | | | | | | 1 | `subi   dx = dy,ui12` | | | | | | | | | |
| | | | | | | | | 2 | `addid  dx = dy,ui12` | | | | | | | | | |
| | | | | | | | | 3 | `subid  dx = dy,ui12` | | | | | | | | | |
| | | | | | | | | 4 | `andi   dx = dy,ui12` | | | | | | | | | |
| | | | | | | | | 5 | `andni  dx = dy,ui12` | | | | | | | | | |
| | | | | | | | | 6 | `ori    dx = dy,ui12` | | | | | | | | | |
| | | | | | | | | 7 | `xori   dx = dy,ui12` | | | | | | | | | |
| | | alu 2 | | ui12/si12 | | | | 0 | `sltiu  dx = dy,ui12` | dy | # | – | – | – | dx | D | – | – |
| | | | | | | | | 1 | `sltis  dx = dy,si12` | | | | | | | | | |
| | | | | | | | | 4 | `muliu  dx = dy,ui12` | | | | | | | | | |
| | | | | | | | | 5 | `mulis  dx = dy,si12` | | | | | | | | | |
| | | alu 3 | | ui20 | | | | | `lui    dx = ui20` | – | # | – | – | – | dx | D | – | – |
| n | = | pgu 4 | x\f,r | $\delta15_H$ | | | y\r | $\delta15_L$ | `alc    px    = dy,δ15` | dy | # | ar | | | px | Pα | – | – |
| m | | | x=f! | $\pi9 : \delta11$ | | | | | `alct   frame = π9,δ11` | # | dr | px | αx | – | px | | | |
| n | | pgu 5 | x\f,r | $\pi15_H$ | | z\r | $\pi15_L$ | | `alc    px    = π15,dz` | # | dz | ar | – | – | px | | | |
| m | | | x=f! | $\pi9 : \delta11$ | | | | | `alcg   frame = π9,δ11` | # | dr | px | αx | – | px | | | |
| n | | pgu 6 | x\f,r | | | | | | `alc    px    = π9,δ11` | # | – | ar | – | – | px | | | |
| m | | | x=f! | | | | | | `alc    frame = π9,δ11` | # | dr | px | αx | – | px | | | |
| af/p | = | lsu 8 | x$^{45}$ | ix12 | | | y\r | 0,1 | `lbu/s  dy = px[ix12]` | – | $dr^7$ | px | αx | – | dy | – | D | – |
| | | | | | | | y\r | 2,3 | `lhu/s  dy = px[ix12]` | | | | | | | | | |
| | | | | | | | y$^1$ | 4,5 | `lwu/s  dy = px[ix12]` | | | | | | | | | |
| | | | | | | | y$^1$ | 7 | `ld     dy = px[ix12]` | | | | | | | | | |
| af/p | | lsu 9 | x$^{45}$ | ix12 | | | y\r | 0 | `sb     px[ix12] = dy` | dy | $dr^7$ | px | αx | – | – | – | – | – |
| | | | | | | | y\r | 1 | `sh     px[ix12] = dy` | | | | | | | | | |
| | | | | | | | y$^1$ | 2 | `sw     px[ix12] = dy` | | | | | | | | | |
| | | | | | | | y$^1$ | 3 | `sd     px[ix12] = dy` | | | | | | | | | |
| af/p | | lsu 9 | x\0$^5$ | ix12 | | | y$^{13}$ | 4 | `lp     py = px[ix12]` | – | $dr^7$ | px | αx | – | py | – | P | α |
| | | | | | | | | 5 | `lcp    py = px[ix12]` | | | | | null | py | | P | α |
| | | | | | | | | 6 | `sp     px[ix12] = py` | | | | | py | – | | – | – |
| | | | | | | | | 7 | `scp    px[ix12] = py` | | | | | py | py | | null | (α) |
| n/p | | lsu A | x\f$^{45}$ | s3 | ud4 | z\r | y\r | 0,1 | `lbu/s  dy = px[dz*s3+ud4]` | – | dz | px | αx | – | dy | – | D | – |
| n/p | | | x\f$^{45}$ | | | z\r | y\r | 2,3 | `lhu/s  dy = px[dz*s3+ud4]` | | | | | | | | | |
| n/p | | | x\f$^{45}$ | | | z\r | y$^1$ | 4,5 | `lwu/s  dy = px[dz*s3+ud4]` | | | | | | | | | |
| a | | | x=f! | | | z=0! | y=r! | 5 | `rst    rix` | | | | | | | | | |
| n/p | | | x\f$^{45}$ | | | z\r | y$^1$ | 7 | `ld     dy = px[dz*s3+ud4]` | | | | | | | | | |
| n/p | | lsu B | x\f$^{45}$ | | | z\r | y\r | 0 | `sb     px[dz*s3+ud4] = dy` | dy | dz | px | αx | – | – | – | – | – |
| n/p | | | x\f$^{45}$ | | | z\r | y\r | 1 | `sh     px[dz*s3+ud4] = dy` | | | | | | | | | |
| n/p | | | x\f$^{45}$ | | | z\r | y$^1$ | 2 | `sw     px[dz*s3+ud4] = dy` | | | | | | | | | |
| a | | | x=f! | | | z=0! | y=r! | 2 | `sv     rix` | | | | | | | | | |
| n/p | | | x\f$^{45}$ | | | z\r | y$^1$ | 3 | `sd     px[dz*s3+ud4] = dy` | | | | | | | | | |
| n/p | | lsu B | x\0,f$^5$ | | | z\r | y\f$^1$ | 4 | `lp     py = px[dz*s3+ud4]` | – | dz | px | αx | – | py | – | P | α |
| a | | | x=f! | | | z=0! | y=r! | 4 | `rst    rcd` | | | | | – | py | | P | α |
| n/p | | | x\0,f$^5$ | | | z\r | y\f$^1$ | 5 | `lcp    py = px[dz*s3+ud4]` | | | | | null | py | | P | α |
| n/p | | | x\0,f$^5$ | | | z\r | y\f$^1$ | 6 | `sp     px[dz*s3+ud4] = py` | | | | | py | – | | – | – |
| a | | | x=f! | | | z=0! | y=r! | 6 | `sv     rcd` | | | | | py | – | | – | – |
| n/p | | | x\0,f$^5$ | | | z\r | y\f$^1$ | 7 | `scp    px[dz*s3+ud4] = py` | | | | | py | py | | null | (α) |
| n | = | bpu C | x\f,r | sd12 | | | y\f,r | 0 | `beqp   px,py,sd12` | – | – | px | – | py | – | – | – | – |
| | | | | | | | | 1 | `bnep   px,py,sd12` | | | | | | | | | |
| | | | $sd12_L$ | $sd12_H$ | | z\r | y\r | 2 | `beq    dy,dz,sd12` | dx | dy | – | – | – | – | – | – | – |
| | | | | | | | | 3 | `bne    dy,dz,sd12` | | | | | | | | | |
| | | | | | | | | 4 | `bgeu   dy,dz,sd12` | | | | | | | | | |
| | | | | | | | | 5 | `bges   dy,dz,sd12` | | | | | | | | | |
| | | | | | | | | 6 | `bltu   dy,dz,sd12` | | | | | | | | | |
| | | | | | | | | 7 | `blts   dy,dz,sd12` | | | | | | | | | |
| n | | bpu D | sd25 | | | | | | `bra    sd25` | – | – | – | – | – | – | – | – | – |
| m | | bpu E | | | | | | | `bsr    sd25` | – | dr | – | – | pf | dr | D | – | – |
| m | | bpu F | x\0,f$^8$ | ix20 | | | | | `jlib   px,ix20` | # | dr | px | αx | pf | dr | D | – | – |

*Abbreviations*

| | |
|---|---|
| n | normal |
| p | privileged |
| a | state aware |
| af | state aware if *px* = *frame* |
| m | state modifying |
| x, y, z | any register number {0, 1, ..., 31} |
| f | 30 (*frame*) |
| r | 31 (*rix*, *rcd, root, core*) |
| = | reserved, should be all zeros |

| | |
|---|---|
| \f | except register number 30 |
| \r | except register number 31 |
| \f,r | except register number 30, 31 |
| \0,r | except register number 0, 31 |
| \0,f | except register numner 0, 30 |
| \0,f,r | except register number 0, 30, 31 |
| =0! | must be register number 0 |
| =f! | must be register number 30 |
| =r! | must be register number 31 |

| | |
|---|---|
| # | immediate |
| dr | d31 = *rix* |
| pf | p30 = *frame* |
| ecd | *x_exc_code_pnt* |
| $\alpha$c | (*x_exc_code_xi*, *x_exc_code_cxi*) |
| eci | *x_exc_code_index* |
| es | *x_exc_status* |
| st | *x_status* |
| | |

*Notes*

[1] if $y = r$: $dy = rix$ or $py = rcd$, instruction privileged

[3] if $y = f$: instruction privileged, *frame* must be loaded from or stored into a stack bumper, illegal if $x = f$ or $ix12 \neq 0$

[4] if $x = 0$: instruction privileged, index always treated as byte index (no implicit scaling)

[5] if $x = r$: $px = root$, instruction privileged, access to root object

[6] if $x = r$ or $y = r$: $px = rcd$ or $py = rcd$, instruction privileged

[7] if $x = f$: instruction state aware

[8] if $x = r$: $px = core$, call routine in *core object*

## LSU/AGU/PGU Instructions

| Instruction | W | EX | ME | AC | addr_mode aux_sel | mem_mode | pgu_mode | load | aload | store. action | astore | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| flshic dz | | | | | dz | flush_ic | – | – | – | flushic | – | S |
| flshdc dz | | | | | dz | flush_dc | – | – | – | flushdc | – | S |
| flshac dz | | | | | – | – | flush | – | – | – | flush | S |
| flshbc dz | – | – | – | – | dz | flush_bc | – | – | – | flushbc | – | S |
| pcce dz,px | | | | | – | – | cxc_put | – | – | – | – | S |
| rcce dz | | | | | – | – | cxc_remove | – | – | – | – | S |
| ccc | | | | | – | – | cxc_clear | – | – | – | – | S |
| **cpp py = px** | py | Pα | | | – | – | cpp | – | – | – | – | – |
| crop py = px | py | Pα | | | – | – | crop | – | – | – | – | – |
| cidp py = px | py | Pα | – | – | – | – | cidp | – | – | – | – | – |
| rpr py = px | py | Pα | | | – | – | rpr | – | – | – | – | – |
| seal px | αx | α! | | | – | – | seal | – | – | – | st_ore | S |
| unsl px | αx | α! | | | – | – | unsl | – | – | – | st_ore | S |
| **alc px = dy,dz** | px | Pα | | – | – | init | alc | – | – | alc | st_ore | S |
| ciop px = dy,dz | px | Pα | | – | – | – | ciop | – | – | – | – | – |
| ccp px = dy | px | P | | α | – | – | ccp | – | l_oad | – | – | L |
| alcb px = dy | px | Pα | – | – | – | init | alcb | – | – | alc | st_ore | S |
| **gcp px** | px | P | | α | – | – | gcp | – | l_oad | – | – | L |
| **dalc frame** | px | P | | α | – | – | dalc_frame | – | l_oad | – | clean | L/S |
| **cpfc rcd** | px | Pα | | – | – | – | cpfc | – | – | – | – | – |
| **alc px = dy,δ15** | px | | | | – | init | alc | – | – | alc | st_ore | S |
| **alct frame = π9,δ11** | **px** | | | | –/delta11 | init_and_clean | alct_frame | – | – | alc | st_ore | S |
| **alc px = π15,dz** | px | Pα | – | – | – | init | alc | – | – | alc | st_ore | S |
| **alcg frame = π9,δ11** | **px** | | | | –/delta11 | init_and_clean | alcg_frame | – | – | alc | st_ore | S |
| **alc px = π9,δ11** | px | | | | –/delta11 | init | alc_aux | – | – | alc | st_ore | S |
| **alc frame = π9,δ11** | **px** | | | | –/delta11 | init_and_clean | alc_frame | – | – | alc | st_ore | S |
| **lbu/s dy = px[ix12]** | | | | | | load | – | true | – | – | – | L |
| **lhu/s dy = px[ix12]** | dy | – | D | – | ix/ix12 | load | – | true | – | – | – | L |
| **lw**u/s **dy = px[ix12]** | | | | | | load | – | true | – | – | – | L |
| ld dy = px[ix12] | | | | | | load | – | true | – | – | – | L |
| **sb px[ix12] = dy** | | | | | | store | – | – | – | st_ore | – | S |
| **sh px[ix12] = dy** | – | – | – | – | ix/ix12 | store | – | – | – | st_ore | – | S |
| **sw px[ix12] = dy** | | | | | | store | – | – | – | st_ore | – | S |
| sd px[ix12] = dy | | | | | | store | – | – | – | st_ore | – | S |
| **lp py = px[ix12]** | py | | P | α | | load | lp | true | l_oad | – | – | L |
| **lcp py = px[ix12]** | py | – | P | α | ix/ix12 | load_and_store | lp | true | l_oad | st_ore | – | L/S |
| **sp px[ix12] = py** | – | | – | – | | store | – | – | – | st_ore | – | S |
| **scp px[ix12] = py** | py | | null | (α) | | store | n_ull | – | – | st_ore | – | S |
| **lbu/s dy = px[dz*s3+ud4]** | | | | | | load | – | true | – | – | – | L |
| **lhu/s dy = px[dz*s3+ud4]** | | | | | | load | – | true | – | – | – | L |
| **lw**u/s **dy = px[dz*s3+ud4]** | dy | – | D | – | dz_s_ud/ud4s3 | load | – | true | – | – | – | L |
| **rst rix** | | | | | | load | – | true | – | – | – | L |
| ld dy = px[dz*s3+ud4] | | | | | | load | – | true | – | – | – | L |
| **sb px[dz*s3+ud4] = dy** | | | | | | store | – | – | – | st_ore | – | S |
| **sh px[dz*s3+ud4] = dy** | | | | | | store | – | – | – | st_ore | – | S |
| **sw px[dz*s3+ud4] = dy** | – | – | – | – | dz_s_ud/ud4s3 | store | – | – | – | st_ore | – | S |
| **sv rix** | | | | | | store | – | – | – | st_ore | – | S |
| sd px[dz*s3+ud4] = dy | | | | | | store | – | – | – | st_ore | – | S |
| **lp py = px[dz*s3+ud4]** | py | | P | α | | load | lp | true | l_oad | – | – | L |
| **rst rcd** | py | | P | α | | load | lp | true | l_oad | – | – | L |
| **lcp py = px[dz*s3+ud4]** | py | – | P | α | dz_s_ud/ud4s3 | load_and_store | lp | true | l_oad | st_ore | – | L/S |
| **sp px[dz*s3+ud4] = py** | – | | – | – | | store | – | – | – | st_ore | – | S |
| **sv rcd** | – | | – | – | | store | – | – | – | st_ore | – | S |
| **scp px[dz*s3+ud4] = py** | py | | null | (α) | | store | n_ull | – | – | st_ore | – | S |

## Compressed Instructions (draft)

| 15 14 13 | 12 11 10 9 8 | 7 6 5 4 3 | 2 1 0 | Instruction | A | B | P | α | Q | W | EX | ME | AC | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 2 | ui5 | y\r | 0 | `addi  dy = ui5` | | | | | | | | | | |
| 2 | ui5 | y\r | 1 | `subi  dy = ui5` | | | | | | | | | | |
| 2 | ui5 | y\r | 2 | `addid dy = ui5` | | | | | | | | | | |
| 2 | ui5 | y\r | 3 | `subid dy = ui5` | | | | | | | | | | |
| 2 | ui5 | y\r | 4 | `andi  dy = ui5` | | | | | | | | | | |
| 2 | ui5 | y\r | 5 | `slli  dy = ui5` | | | | | | | | | | |
| 2 | ui6 | y\r | 6,7 | `sllid dy = ui6` | | | | | | | | | | |
| 3 | z\r | y\r | 0 | `add   dy = dz` | | | | | | | | | | |
| 3 | z\r | y\r | 1 | `sub   dy = dz` | | | | | | | | | | |
| 3 | z\r | y\r | 2 | `addd  dy = dz` | | | | | | | | | | |
| 3 | z\r | y\r | 3 | `subd  dy = dz` | | | | | | | | | | |
| 3 | z\r | y\r | 4 | `and   dy = dz` | | | | | | | | | | |
| 3 | z\r | y\r | 5 | `sll   dy = dz` | | | | | | | | | | |
| 3 | z\r | y\r | 6 | `slld  dy = dz` | | | | | | | | | | |
| 3 | z\r | y\r | 7 | `cp    dy = dz` | | | | | | | | | | |
| 3 | z=r! | y\r | 0 | `pop` | | | | | | | | | | |
| 3 | z=r! | y\r | 1 | `gcp   py` | | | | | | | | | | |
| 3 | z=r! | y\r | 2 | `cpfc  rcd` | | | | | | | | | | |
| 3 | z=r! | y\r | 3 | `check rcd` | | | | | | | | | | |
| 3 | z=r! | y\r | 4 | `jmp   dy` | | | | | | | | | | |
| 3 | z=r! | y\r | 5 | `jsr   dy` | | | | | | | | | | |
| 3 | z=r! | y\r | 6 | `rts` | | | | | | | | | | |
| 3 | z=r! | y\r | 7 | `rtlb` | | | | | | | | | | |
| 4 | ui6 | y\r | 0,1 | `li    dy = ui6` | | | | | | | | | | |
| 4 | ui6 | y\r | 2,3 | `lni   dy = ui6` | | | | | | | | | | |
| 4 | ui6 | y\r | 4,5 | `lui   dy = ui6` | | | | | | | | | | |
| 4 | z\r | y\r | 6 | `not   dy = dz` | | | | | | | | | | |
| 4 | z\r | y\r | 7 | `neg   dy = dz` | | | | | | | | | | |
| 5 | ui5 | y\r | 0 | `rstw  dy,ui5` | | | | | | | | | | |
| 5 | ui5 | y\r | 1 | `svw   dy,ui5` | | | | | | | | | | |
| 5 | ui5 | y\r | 2 | `rstd  dy,ui5` | | | | | | | | | | |
| 5 | ui5 | y\r | 3 | `svd   dy,ui5` | | | | | | | | | | |
| 5 | ui5 | y\r | 4 | `rstp  py,ui5` | | | | | | | | | | |
| 5 | ui5 | y\r | 5 | `svp   py,ui5` | | | | | | | | | | |
| 5 | = | y=r! | 0 | `rstrix` | | | | | | | | | | |
| 5 | = | y=r! | 1 | `svrix` | | | | | | | | | | |
| 5 | = | y=r! | 4 | `rstrcd` | | | | | | | | | | |
| 5 | = | y=r! | 5 | `svrcd` | | | | | | | | | | |
| 5 | z\f | y\f | 6 | `cpp   py = pz` | | | | | | | | | | |
| 5 | pi5 | dt5 | 7 | `push  pi5,dt5` | | | | | | | | | | |
| 6 | ui4 z’ | ui4 y’ | 0 | `lw    dy’ = pz’[ui4]` | | | | | | | | | | |
| 6 | ui4 z’ | ui4 y’ | 1 | `sw    pz’[ui4] = dy’` | | | | | | | | | | |
| 6 | ui4 z’ | ui4 y’ | 2 | `ld    dy’ = pz’[ui4]` | | | | | | | | | | |
| 6 | ui4 z’ | ui4 y’ | 3 | `sd    pz’[ui4] = dy’` | | | | | | | | | | |
| 6 | ui4 z’ | ui4 y’ | 4 | `lp    py’ = pz[ui4]` | | | | | | | | | | |
| 6 | ui4 z’ | ui4 y’ | 5 | `sp    pz’[ui4] = py’` | | | | | | | | | | |
| 6 | ui4 z’ | ui4 y’ | 6 | `--` | | | | | | | | | | |
| 6 | 0 z’ | 0 y’ | 7 | `--` | | | | | | | | | | |
| 6 | 0 z’ | 1 y’ | 7 | `andn  dy’ = dz’` | | | | | | | | | | |
| 6 | 0 z’ | 2 y’ | 7 | `or    dy’ = dz’` | | | | | | | | | | |
| 6 | 0 z’ | 3 y’ | 7 | `xor   dy’ = dz’` | | | | | | | | | | |
| 6 | 1 z’ | 0 y’ | 7 | `srl   dy’ = dz’` | | | | | | | | | | |
| 6 | 1 z’ | 1 y’ | 7 | `sra   dy’ = dz’` | | | | | | | | | | |
| 6 | 1 z’ | 2 y’ | 7 | `srld  dy’ = dz’` | | | | | | | | | | |
| 6 | 1 z’ | 3 y’ | 7 | `srad  dy’ = dz’` | | | | | | | | | | |
| 6 | 4 ui2 | y’ | 7 | `srl   dy = ui2` | | | | | | | | | | |
| 6 | 5 ui2 | y’ | 7 | `sra   dy = ui2` | | | | | | | | | | |
| 6 | 6 ui2 | y’ | 7 | `srld  dy = ui2` | | | | | | | | | | |
| 6 | 7 ui2 | y’ | 7 | `srad  dy = ui2` | | | | | | | | | | |
| 7 | sd8 | 0 y’ | sd8 | `beq   dy’,sd8` | | | | | | | | | | |
| 7 | sd8 | 1 y’ | sd8 | `bne   dy’,sd8` | | | | | | | | | | |
| 7 | sd11 | 2 sd11 | sd11 | `bra   sd11` | | | | | | | | | | |
| 7 | sd11 | 3 sd11 | sd11 | `bsr   sd11` | | | | | | | | | | |

## Pointer and Attribute Encoding

| pointer | | π attribute | | | | δ attribute | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| kind | tags[1] | 31 | 30 ... 2 | 1 | 0 | 31 | 30 | 29 | 28 ... 1 | 0 |
| pointer properties | | object properties | | | | | | | | |
| code[2] | 000 | 1 | $\xi_{29}$ | $00_2$! | | 1 | $\Xi_{30}$[6] | | | priv |
| uninitialized | | 1 | $\pi_{29}$ | gc gen[3] | gc visited[3] | 0 | $\delta_{31}$ | | | |
| ordinary | | 0 | | | | s | | | | |
| read only | 001 | 0! | | | | | | | | |
| id only | 010 | | | | | | | | | |
| io | 011 | don't care[4] | | | | | | | | |
| frame square black | 100 | uini | $\pi_{29(9)}$[5] | bumper bit | gc visited[3] | lib | rc | ri | $\delta_{29(11)}$[5] | |
| frame square white | 101 | | | | | | | | | |
| frame round black | 110 | | | | | | | | | |
| frame round white | 111 | | | | | | | | | |

*Abbreviations*

| | |
|---|---|
| uini | set if initialization is incomplete |
| priv | privilege level (00: user, 11: supervisor, 01/10: reserved) |
| s | seal bit, set to seal object (read only) |
| lib | set to identify library entry stack frames |
| rc | set to reserve pointer at index 0 for *rcd* (if *rc* = 1, *ri* must be 1) |
| ri | set to reserve data word at index 0 for *rix* |

*Notes*

[1] invariant: the null pointer tags must always be 000

[2] pointer to the *core object* is *–8*, must never be dereferenced, attributes are implicit and never loaded

[3] reserved for garbage collection

[4] δ encoded in the pointer, π implicitly 0, attributes are never loaded from or stored to memory

[5] frame objects can only be allocated with static attributes, so only 9/11 bits are actually used

[6] to supprt the "compressed" extension, not implemented yet (currently 29 bits for Ξ and 2 bits for priv)

## Object and Attribute Access regarding Pointer and Object Properties

| | user mode | | | | supervisor mode | | | |
|---|---|---|---|---|---|---|---|---|
| | load | store | qpi | qdtx[1] | load | store | qpi | qdtx[1] |
| code | ***drfcd*** | ***drfcd*** | ***drfcd*** | ***drfcd*** | yes | yes | $\xi_{29}$00 >> 2 | $\Xi_{29}$00 >> *scl*[1] |
| uini | ***panic*** | ***panic*** | ***panic*** | ***panic*** | ***panic*** | ***panic*** | $\pi_{29}$00 >> 2 | $\delta_{31}$ >> *scl*[1] |
| ordinary[3] | yes | yes[5] | $\pi_{29}$00 >> 2 | $\delta_{31}$ >> *scl*[1] | yes | yes | $\pi_{29}$00 >> 2 | $\delta_{31}$ >> *scl*[1] |
| read only[3,4] | yes | ***wrptv*** | $\pi_{29}$00 >> 2 | $\delta_{31}$ >> *scl*[1] | yes | yes | $\pi_{29}$00 >> 2 | $\delta_{31}$ >> *scl*[1] |
| id only[3] | ***drfid*** | ***drfid*** | ***drfid*** | ***drfid*** | yes | yes | $\pi_{29}$00 >> 2 | $\delta_{31}$ >> *scl*[1] |
| io | yes | yes | 0 | $\delta_{16}$[2] >> *scl*[1] | yes | yes | 0 | $\delta_{16}$[2] >> *scl*[1] |
| frame | yes[6] | yes[6] | $\pi_{29}$00 >> 2[6] | $\delta_{29}$ >> *scl*[1,6] | yes[6] | yes[6] | $\pi_{29}$00 >> 2[6] | $\delta_{29}$ >> *scl*[1,6] |

[1] qdtx = *qdtb* (= *qdt*), *qdth*, *qdtw* or *qdtd*; *scl* = 0 for *qdtb*, *scl* = 1 for *qdth*, *scl* = 2 for *qdtw*, *scl* = 3 for *qdtd*

[2] for register capabilities: $\delta_{16}$ >> *scl*, for memory capabilities: ($\delta_{16}$ << 12) >> *scl*

[3] pointers read from *sealed* objects in user mode are set to *read only* (pointers to *code*, *read only*, *id only* and *io* objects remain unmodified, pointers to *uini* or *frame* objects will cause panics anyway)

[4] pointers read from *read only* objects in user mode are set to *read only* (pointers to *code*, *read only*, *id only* and *io* objects remain unmodified, pointers to *uini* or *frame* objects will cause panics anyway)

[5] ***sealv*** if sealed

[6] only by dereferencing *p30* = *frame* and unless in a quarantine state, access restrictions to index 0 apply

## Decoding *load dy/py = px[index]* and *store px[index] = dy/py* instructions

| px | dy/py | index | | |
|---|---|---|---|---|
| null | py | | illegal | physical memory accesses are always data accesses |
| | rix | | illegal | makes no sense |
| | else | | privileged | physical memory access: no implicit scaling, unaligned: ***sverr*** |
| frame | | dyn | illegal | impossible to implement (because of state checking) |
| | frame | stat | illegal | frame must be saved/restored to/from a bumper not referred by *frame* (why?) |
| | rix, rcd | stat ≠ 0 | privileged | save/restore rcd, rcd to/from process state |
| | rix, rcd | stat 0 | normal | save/restore rix, rcd to/from stack frame |
| | else | stat | normal | access stack frame |
| rcd/root | frame | | illegal | frame must be saved/restored to/from a bumper |
| | else | | privileged | root object access |
| else | frame | dyn | illegal | frame must be saved/restored to/from a bumper at static index 0 |
| | | stat ≠ 0 | illegal | frame must be saved/restored to/from a bumper at static index 0 |
| | | stat 0 | privileged | save/restore frame to/from bumper (bumper bit checked dynamically) |
| | rix, rcd | | privileged | save/restore rix, rcd to/from process state |
| | else | | normal | ordinary object access |

*Notes*
- loading *rix* with *lb* or *lh* instructions is illegal, *rix* must be loaded with a *lw* instruction
- storing *rix* with *sb* or *sh* instructions is illegal, *rix* must be stored with a *sw* instruction

## Sanity check for pointers and attributes

required input: pointer [P], attributes [A], register number [R], privilege mode [M], index [I]

*Dereferenced pointer sanity check*

```
if $p_{31..3} = 0$ and $p_{2..0} \neq 000$ then panic [P] [deref irregular null]
switch $p_2$
    case 0: if reg_no = frame then panic [PR] [deref non-frame pointer in frame]
    case 1: if reg_no ≠ frame and mode = user then panic [PRM]
        [deref frame pointer in non-frame register visible in user mode]
        [supervisor may deref a pointer to a bumper in a non-frame register (only)]
```

*Loaded/Stored pointer sanity check (lp/lcp/sp/scp only)*

```
if $p_{31..3} = 0$ and $p_{2..0} \neq 000$ then panic [P] [load/store irregular null]
switch $p_2$
    case 0: if reg_no = frame then panic [PR] [load/read non-frame pointer to/from frame]
    case 1: if mode = user then panic [PM] [load/read frame pointer in user mode, no matter to what register]
```

*Dereferenced attributes sanity check*

```
if $\pi_{31} = 1$ then
    switch $p_{2..0}$
        case 000:
            switch $\delta_{31}$
                case 0: panic [PA] [access uninitialized ordinary]
                case 1: if $\pi_{1..0} \neq 00$ or $\delta_{1..0} \neq 00 \mid 11$ then panic [PA] [access irregular code]
        case 001: panic [PA] [access uninitialized read only]
        case 010: panic [PA] [access uninitialized id only]
        case 1xx: panic [PA] [access uninitialized frame]
if $p_2 = 1$ then
    (assert reg_no = frame or mode = super)
    if $\pi_{30..11} \neq 0$ or $\delta_{28..11} \neq 0$ or ($\delta_{30} = 1$ and $\delta_{29} = 0$) then panic [PA] [access irregular frame]
    switch $\pi_1$
        case 0:
            if reg_no ≠ frame then panic [PAR] [access ordinary frame via non-frame register]
        case 1:
            if mode = user then panic [PAM] [access bumper in user mode]
            if $\pi_{30..2} \neq 1$ or $\delta_{28..0} \neq 0$ or $\pi_{31} = 1$ or $\delta_{31..29} \neq 000$ then panic [PA] [access irregular bumper]
(rcd cannot be checked because rcd is replaced by root when dereferenced)
```

*Loaded attributes sanity check (lp/lcp only)*

```
if $\pi_{31} = 1$ then
    switch $p_{2..0}$
        case 000:
            switch $\delta_{31}$
                case 0: if mode = user then panic [PAM] [load pointer to uninitialized ordinary in user mode]
                case 1: if $\pi_{1..0} \neq 00$ or $\delta_{1..0} \neq 00 \mid 11$ then panic [PA] [load pointer to irregular code]
        case 001: panic [PA] [load pointer to uninitialized read only]
        case 010: panic [PA] [load pointer to uninitialized id only]
        case 1xx: if mode = user then panic [PAM] [loaded pointer to uninitialized frame in user mode]
if $p_2 = 1$ then
    (assert mode = super)
    if $\pi_{30..11} \neq 0$ or $\delta_{28..11} \neq 0$ or ($\delta_{30} = 1$ and $\delta_{29} = 0$) then panic [PA] [load pointer to irregular frame]
    switch $\pi_1$
        case 0:
            if reg_no ≠ frame then panic [PAR] [load pointer to ordinary frame to non-frame register]
        case 1:
            if $\pi_{30..2} \neq 1$ or $\delta_{28..0} \neq 0$ or $\pi_{31} = 1$ or $\delta_{31..29} \neq 000$ then panic [PA] [load pointer to irregular bumper]
if reg_no = rcd and $p_{31..3} \neq 0$ and ($p_{2..0} \neq 000$ or $\pi_{31} \neq 1$ or $\delta_{31} \neq 1$) then panic [PAR]
    [load pointer to non-code object to rcd]
```

## Checking *load dy/py = px[index]* and *store px[index] = dy/py* instructions

```
if illegal then illeg

if privileged and mode = user then privv

call dereferenced pointer sanity check for px

if instr = sp then call stored pointer sanity check for py

switch $px_{2..0}$
    case 000: if $p_{31..3}$ = 0 then drfnu [P]
    case 001: if mode = user and instr = sb | sh | sw | sp then wrptv [PM]
    case 010: if mode = user then drfid [PM]

if $px_2$ = 1 and state = quarantine then sterr [PS]

call dereferenced attribute sanity check for px

if $px_{2..0}$ = 000 then
    if $\delta_{31}$ = 1 and $\pi_{31}$ = 1 then
        switch mode
            case user: drfcd [PAM]
            case super: if instr = lp | lcp | sp | scp then sverr [PAM]

if $px_{2..0}$ = 000 | 001 | 010 and $\delta_{31}$ = 0 and $\pi_{31}$ = 1 and instr = sb | sh | sw | sp and mode = user then sealv [PAM]

if $px_2$ = 1 and $\pi_1$ = 1 then
    (assert mode = super)
    if reg_no_x = frame then sverr [PAR] [access bumper via frame register]
    if reg_no_y ≠ frame then sverr [PAR] [store something other than frame to bumper]
else
    if reg_no_y = frame then (assert mode = super) sverr [PAR] [load/store frame from/to non-bumper]

switch reg_no_x
    case null:
        (assert mode = super)
        switch instr
            case lb | sb:
            case lh | sh: if $index_0$ ≠ 0 then sverr [I] [alignment error]
            case lw | sw: if $index_{1..0}$ ≠ 00 then sverr [I] [alignment error]
            case lp | lcp | sp | scp: (assert false) [illegal]
    case not null:
        if carry during index calculation or index out of bounds then ixoob [AI]

if py = rcd and mode = user and $\delta_{30}$ = 0 then fram0 [AM]
if dy = rix and mode = user and $\delta_{29}$ = 0 then fram0 [AM]

if px = frame then
    switch instr
        case lb | sb: if index < 4 and $\delta_{29}$ = 1 then fram0 [IA]
        case lh | sh: if index < 2 and $\delta_{29}$ = 1 then fram0 [IA]
        case lw | sw: if dy ≠ rix and index < 1 and $\delta_{29}$ = 1 then fram0 [IA]
        case lp | lcp | sp | scp: if py ≠ rcd and index < 1 and $\delta_{30}$ = 1 then fram0 [IA]

if instr = lp | lcp then
    call loaded pointer sanity check for py
    call loaded attributes sanity check for py
```

Rules and invariants

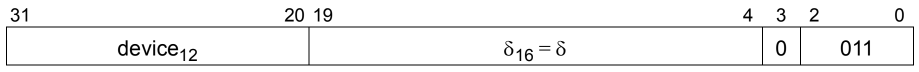
- *frame* exclusively holds frame pointers (supervisor has to ensure that frame is written by *alcb* before it is read)
- if *frame* refers to a bumper, it may not be dereferenced
- a frame bumper may only be dereferenced in a non-*frame* register
- ordinary pointer registers never hold frame pointers in user mode
- ordinary pointer registers may hold pointers to frame bumpers in supervisor mode (but not to ordinary frames)
- pointers to ordinary frames may exclusively be held in *frame* or stored in a bumper

pointer generating instructions: *alc*, *alct*, *alcg*, *alcb*, *lp*, *lcp, scp*, *lssp*, *cpp*, *crop*, *cidp*, *rpr*, *seal*, *unsl*, *ciop*, *ccp*, *gcp*, *dalc*, *cpfc*

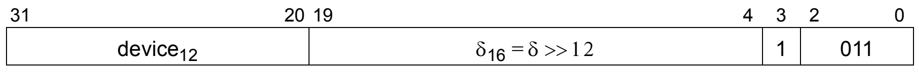
## IO Pointer Encoding

The attributes of an io pointer are derived from the pointer value as follows:

„register capability“: $0 < \delta \leq \$0000FFFF$, byte granularity ($\delta_{31..16} = 0!$)

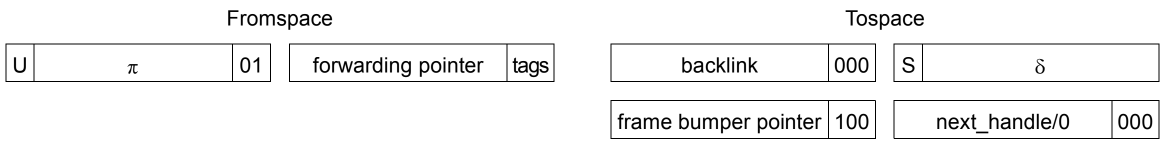


„memory capability“: $0 < \delta \leq \$0FFFF000$, 4k granularity ($\delta_{31..28} = 0!$, $\delta_{11..0} = 0!$)



The $\pi$ attribute of an io pointer is always and implicitly 0.

## Attribute Tags during Garbage Collection



**Special pointer registers**

| | | read | write | deref for load/store | deref to query attributes |
|---|---|---|---|---|---|
| p0 | u: null | yes: null pointer | ignored | privilege violation fault | illegal instruction fault |
| | s: null/mem | yes: null pointer | ignored | phys. memory (data only) | illegal instruction fault |
| p30 | u: frame | no | only alc, dalc | yes (unless in quarantine) | yes (unless in quarantine) |
| | s: frame | only by store to bumper | only alc, dalc or by load from bumper | yes (unless in quarantine) | yes (unless in quarantine) |
| p31 | u: rcd, core | only by sp/scp frame[0] := rcd | only by cpfc rcd or by lp/lcp rcd := frame[0] | illegal instruction fault | illegal instruction fault |
| | s: rcd, core, root | only by store instructions | only by cpfc rcd or by load instructions | yes (access root object) | yes (access root object) |

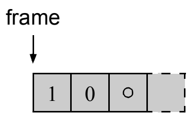
**Special data registers**

| | | read | write |
|---|---|---|---|
| d0 | u: zero | yes: value 0 | ignored |
| | s: zero | yes: value 0 | ignored |
| d31 | u: rix | only by sw frame[0] := rix | only by lw rix := frame[0] |
| | s: rix | only by sw instructions | only by lw instructions |

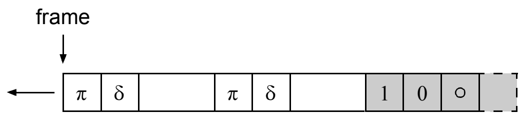
## Stack

allocate stack bumper (end of stack) (privileged)
stack bumper initially contains a null pointer to indicate that the stack is active (i.e. referred to by frame)

```
alcb frame := addr
```

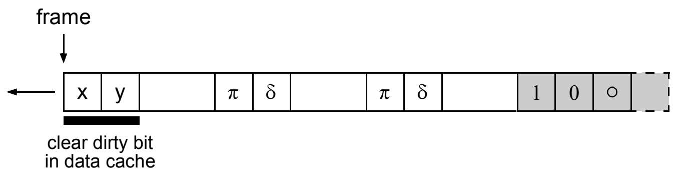


Stack structure with two stack frames



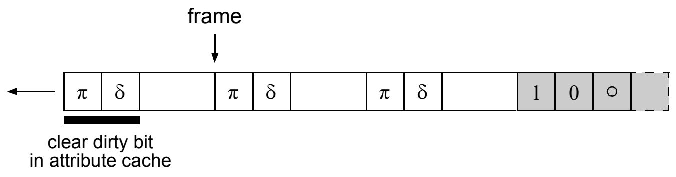
frame allocate

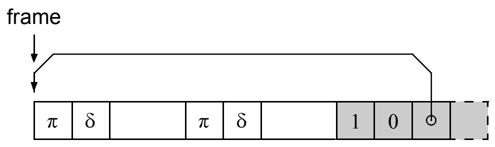
```
alc frame := #x,#y
```



frame deallocate

```
dalc frame
```





Deactivate a stack: write frame to the bumper (privileged)

```
sw pb[0] := frame
```

Activate a stack: load the top frame pointer from the bumper, clear the pointer in the bumper (privileged)

```
lssp frame := pb[0]
```

### Invariants

- at any given time, there is at most one active stack whose top end is referred to by *frame*
- if *frame* refers to a stack bumper, the corresponding stack is empty, i.e. frame may never refer to the bumper of a deactivated stack
- at any given time, if a stack bumper contains *null*, then *frame* points to the top frame of the corresponding stack (*frame* may also point to the top frame of an inactive stack)
- at any given time, the bumper of the stack referred to by frame (the active stack) contains *null* (frame may however refer to an inactive stack)
- not true: the active stack always contains a null pointer in its bumper, an inactive stack always contains a pointer to the top stack frame

### Cache coherence challenges

- frame allocate has to remove the dirty tag in the data cache at the new attribute address
(avoid dead dirty data overwrite attributes)
- frame deallocate has to remove the dirty tag from the corresponding attribute cache entry
(avoid dead dirty attributes overwrite data)

Ordinary Stack Frames
– implicitly allocated by alc frame := ... in quarantine states QC and QU
– possible to store rix
Gate Stack Frames
– explicitly allocated by alcg frame := in any quarantine state
– possible to store rix and rcd
Terminal Stack Frames
– explicitly allocated by alct frame := in any quarantine state
– impossible to store rix and rcd
Library Stack Frames
– orthogonal to Terminal and Gate Frames
– implicitly allocated by alc/alct/alcg frame := ... in state QL

| alct frame (rc = 0, ri = 0) | alc frame (rc = 0, ri = 1) | alcg frame (rc = 1, ri = 1) |
|---|---|---|
| $\pi + \delta >= 0$; ***aperr*** otherwise | $(\pi + \delta >= 0)$, $\delta >= 4$; ***aperr*** otherwise | $(\pi + \delta >= 0)$, $\pi >= 1$, $\delta >= 4$; ***aperr*** otherwise |

**Terminal subroutine (containing no intra/inter code object calls)**

```
alct frame := #x,#y
...
dalc frame
rts
```

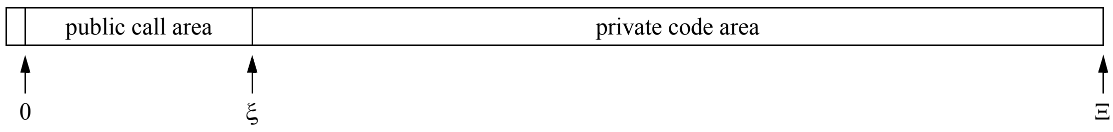
**Standard subroutine (containing intra code object calls only)**

```
alc frame := #x,#y
sw frame[0] := rix
...
lw rix := frame[0]
dalc frame
rts
```

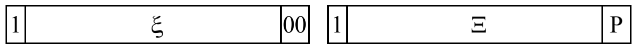
**Gate subroutine (containing inter code object calls or intra code object calls that contain inter code object calls)**

```
alcg frame := #x,#y
sp frame[0] := rcd
sw frame[0] := rix
cpfc rcd
check
...
unchk <==== DEPRECATED
lw rix := frame[0]
lp rcd := frame[0]
dalc frame
rtlb
```

**Code objects**



Attribute encoding for code objects



| | |
|---|---|
| ξ | size of public call area in number of words |
| Ξ | size of code object in words |
| P | privilege level |

– code objects are used for the kernel, kernel modules, libraries and application programs

– privileged code objects contain operating system code, user code objects contain application programs and libraries

– code objects are identified by $\pi[31] = 1$ and $\delta[31] = 1$

– privilege level encoded in lower two bits of the code object's δ attribute (00 = user, 11 = supervisor, 01/10 = reserved)

– there is one distinguished *core object* at physical address –8
  – contains the interrupt, fault and trap handlers; *x_intv*, *x_faultv*, *x_trapv* are indexes into the super code object
  – may exclusively execute the instruction *rte* (*sverr* otherwise)
  – deprecated: a pointer to the super code object is identical to the *null* pointer, may never be dereferenced
  – deprecated: may not call code in other code objects (!!!) (*rcd* would be *null*)
  – the attributes of the *core object* are implicit (*x_core_xi*, *x_core_cxi*) and never loaded from memory
  – deprecated: ξ of the super code object is 0, code in the super code object is never called by *jlib*
  – deprecated: code in the super code object is exclusively called by traps, faults, interrupts and reset
  – deprecated: consequently, the contents of first 8 bytes of the physical address space are „don't care"

– privileged instruction *ccp* creates a pointer to a code object

– code pointer not part of pointer register file (like pc is not part of the data register file)

– argument of *jmp*, *jsr* (and *rts*, *rtlb*): index into current code object

– *jlib px,index* to call subroutines in other code objects
  – *px* must be a code pointer different from a pointer to the current code object
  – *index* must refer to the first instruction of a subroutine within the public call area of a code object referred to by *px*
  – copies the return index to *rix*
  – only allowed in state RC (state assures that *code* = *rcd*)

– *rtlb* returns to calling code object
  – *rcd* must contain a return code pointer different from a pointer to the current code object
  – only allowed in state QL (state assures that the library stack frame has been deallocated)

– architecturally supported destruction (and relocation?) of code objects: garbage collector invalidates obsolete code pointers, detects references to the code object to be destroyed (converts them to *null* pointers) (or relocated?)

– index into code object, length of code object counted in words, not bytes

– user mode instruction *gcp px* to obtain a pointer to a context object associated with the current code object

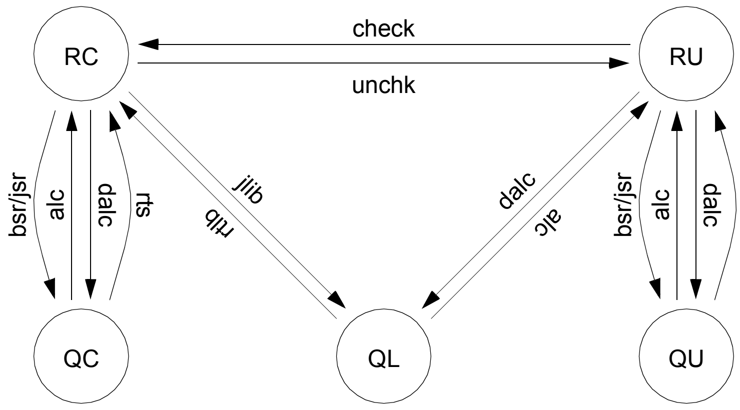
– *bra*, *bcc*, *bsr*, *jsr* and *jmp* verify that the target code index is within the current code object, ***tciob*** otherwise

– *jlib* verifies that the target code index is within the target code object, ***tciob*** otherwise

– *jlib* verifies that the target code object is a code object different from the current code object and not *null*, ***tcoil*** otherwise

– *rts* and *rtlb* do not check *rix*, instruction after corresponding *bsr*/*jsr* or *jlib* will cause ***endoc***

**frame, rix and rcd protection (partially deprecated, especially unchk)**



State encoding

| | | frame shape = rix shape? | frame color = rix color? | unchecked bit set? | invariant |
|---|---|---|---|---|---|
| RC | Regular Checked | yes | yes | no | rcd = code |
| RU | Regular Unchecked | yes | yes | yes | |
| QC | Quarantine Checked | yes | no | no | rcd = code |
| QU | Quarantine Unchecked | yes | no | yes | |
| QL | Quarantine Library | no | no | yes | |

State transitions

| | src | tgt | rix | | | | frame | | further conditions or actions |
|---|---|---|---|---|---|---|---|---|---|
| | | | unchecked$_{31}$ | shape$_{30}$ | color$_{29}$ | value$_{28..0}$ | shape$_1$ | color$_0$ | |
| alc frame | QC<br>QU<br>QL | RC<br>RU<br>RU | | | | | keep<br>keep<br>toggle | toggle<br>toggle<br>toggle | allocate ordinary frame<br>allocate ordinary frame<br>allocate library entry frame |
| dalc frame | RC<br>RU<br>RU | QC<br>QU<br>QL | | | | | keep<br>keep<br>toggle | toggle<br>toggle<br>toggle | verify that frame is ordinary<br>if frame is ordinary<br>if frame is library entry |
| bsr/jsr | RC<br>RU | QC<br>QU | keep | keep | toggle | set | | | |
| rts | QC<br>QU | RC<br>RU | keep | keep | toggle | clear | | | verify that rix != 0 |
| jlib | RC | QL | set | toggle | toggle | set | | | verify that target != code, null<br>and that target is code object |
| rtlb | QL | RC | clear | toggle | toggle | clear | | | verify that rcd != code, null<br>and that target is code object<br>verify that rix != 0 |
| unchk | RC | RU | set | keep | keep | keep | | | |
| check | RU | RC | clear | keep | keep | keep | | | verify that rcd = code |
| lw rix :=<br>frame[0] | RC<br>RU | RC<br>RU | ***sterr*** if bit changes | ***panic*** if bit changes | ***panic*** if bit changes | set, ***rixeq*** if value=0 | | | verify that value != 0 and that no implicit state transition occurs |

State privileges

| | | bsr/jsr | rts | jlib | rtlb | alc frame | dalc frame | check | unchk | access frame, query frame attributes | restore rcd |
|---|---|---|---|---|---|---|---|---|---|---|---|
| RC | Regular Checked | yes | no | yes | no | no | yes | no! | yes | yes | no |
| RU | Regular Unchecked | yes | no | no | no | no | yes | yes | no! | yes | yes |
| QC | Quarantine Checked | no | yes | no | no | yes | no | no | no | no | impossible |
| QU | Quarantine Unchecked | no | yes | no | no | yes | no | no | no | no | impossible |
| LQ | Library Quarantine | no | no | no | yes | yes | no | no | no | no | impossible |

State modifying instructions

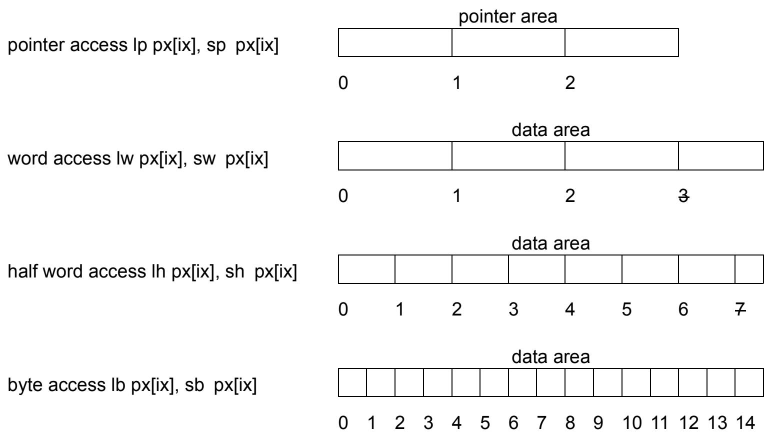
| | read a | read b | read p | read pα | read q | write |
|---|---|---|---|---|---|---|
| bsr<br>jsr | *rix (st)* | –<br>target index | | | *frame (st)* | *rix (ri, st)* |
| rts | *rix (st)* | rix (ri) | | | *frame (st)* | *rix (ri, st)* |
| jlib px.ix<br>jlib px,dz | *rix (st)* | –<br>target index | target code | target code | *frame (st)* | *rix (ri, st)* |
| rtlb | *rix (st)* | rix (ri) | rcd | rcd | *frame (st)* | *rix (ri, st)* |
| check | *rix (st)* | | rcd | | *frame (st)* | *rix (st)* |
| unchk | *rix (st)* | | | | *frame (st)* | *rix (st)* |
| alc frame | *rix (st)* | | frame | frame | | frame |
| dalc frame | *rix (st)* | | frame | frame | | frame |

State aware instructions

| | read a | read b | read p | read pα | read q | write |
|---|---|---|---|---|---|---|
| lw dy := frame[d]<br>lp/lcp py := frame[d] | | *rix (st)* | frame | frame | | dy<br>py |
| sw frame[d] := dy<br>sp/scp frame[d] := py | dy<br>– | *rix (st)* | frame | frame | –<br>py | |
| query attributes | | *rix (st)* | frame | frame | | dy |

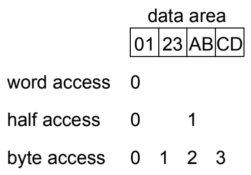
**Object Access**

Example: Object with $\pi = 3$, $\delta = 15$


pointer area
pointer access lp px[ix], sp px[ix]
0 1 2
data area
word access lw px[ix], sw px[ix]
0 1 2 ~~3~~
data area
half word access lh px[ix], sh px[ix]
0 1 2 3 4 5 6 ~~7~~
data area
byte access lb px[ix], sb px[ix]
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14

Words and half words are stored in big endian format. Example (pseudo code)

```
sw p4[0] := $0123ABCD
lw  d4 := p4[0]   ;d4 = $0123ABCD
lhu d4 := p4[0]   ;d4 = $00000123
lhu d4 := p4[1]   ;d4 = $0000ABCD
lbu d4 := p4[0]   ;d4 = $00000001
lbu d4 := p4[1]   ;d4 = $00000023
lbu d4 := p4[2]   ;d4 = $000000AB
lbu d4 := p4[3]   ;d4 = $000000CD
lhs d4 := p4[0]   ;d4 = $00000123
lhs d4 := p4[1]   ;d4 = $FFFFABCD
lbs d4 := p4[0]   ;d4 = $00000001
lbs d4 := p4[1]   ;d4 = $00000023
lbs d4 := p4[2]   ;d4 = $FFFFFFAB
lbs d4 := p4[3]   ;d4 = $FFFFFFCD
```



Fault priorities

| | |
|---|---|
| 1 Pointer related | 1.1 ***drfnu***: deref null |
| | 1.2 ***drfid***: deref id; ***wrptv***: write protection violation |
| 2 Attribute related | 2.0 ***panic***: invalid frame attributes (rc = 1, ri = 0) |
| | 2.1 ***drfui***, ***drfcd***, ***drfbp***, ***sealv***: deref uninitialized, deref code, dref bumper, seal violation |
| | 2.2 ***sterr***: state error |
| | 2.3 ***ixoob***: index out of bounds |
| | 2.4 ***fram0***: frame[0] access (see below) |

| | | sw frame[0] := dy<br>lw dy:= frame[0] | | sp/scp frame[0] := py<br>lp/lcp py := frame[0] | |
|---|---|---|---|---|---|
| rc | ri | dy = rix | dy != rix | py = rcd | py != rcd |
| 0 | 0 | ***fram0*** | ok | ***fram0*** | ok |
| 0 | 1 | ok | ***fram0*** | ***fram0*** | ok |
| 1 | 1 | ok | ***fram0*** | ok | ***fram0*** |
| 1 | 0 | ***panic*** (not used) | | | |

**Privileged instructions detailed**

*ctsr xy := dz*, *cfsr dz := xy*, *sync*
no faults (except illegal)

*load dy := mem[addr], store mem[addr] := dy*
*addr* measured in bytes, ***sverr*** for misaligned operands or pointer access

*pcce dz,px*, *rcce dz*, *flushic dy*, *flushdc dy*, *flushac dy*, *flushbc dy*
no faults (except illegal)

*ccp px := dz*
loads attributes from memory; verifies that π[31] and δ[31] are set, that π[1..0] are cleared, that that δ[0] = δ[1], that *dz* is a multiple of 8 and that *dz* != 0, ***sverr*** otherwise

*alcb px := dz*
creates a frame square white pointer (initially, *rix* = 0, so state will be *QC*); writes attributes to memory (clears *uini*, *gc*, *lib*, *rc*, *ri*); verifies that *dz* is a multiple of eight and *dz* != 0, ***sverr*** otherwise

*ciop px := dy,dz*
creates an io pointer with device = *dy*, δ = *dz*, ***sverr*** if $\text{device}_{31..12}$ != 0
„register capability“ if $\delta < 2^{16}$, „memory capability“ if $\delta \geq 2^{16}$
***sverr*** if δ = 0 or $\delta_{31..28}$ != 0 or ($\delta_{27..16}$ != 0 and $\delta_{11..0}$ != 0)

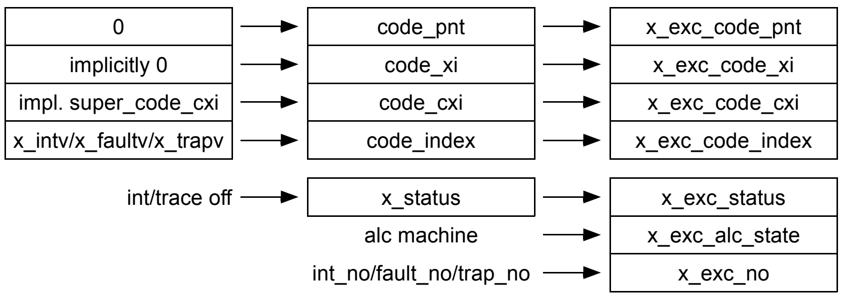
*qptr dy := px*
returns raw pointer including tags; no faults

*qpir dy := px*, *qdtr dy := px*
returns raw attributes as physically stored in the registers; **no faults**

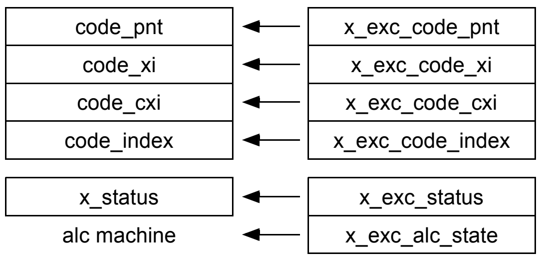
*crop py := px*, *cidp py := px*, *rpr py := px*, *seal px*, *unsl px*
see table below

| | | crop | cidp | rpr | seal | unsl |
|---|---|---|---|---|---|---|
| null | – | ***sverr*** | | | | |
| ordinary | code | ***sverr*** | | | | |
| | uini | ***sverr*** | | | | |
| | sealed | rd only - sealed | id only - sealed | ordinary - sealed | ordinary - sealed | ordinary - unsealed |
| | unsealed | rd only - unsealed | id only - unsealed | ordinary - unsealed | ordinary - sealed | ordinary - unsealed |
| rd only | code/uini | ***sverr*** | | | | |
| | sealed | rd only - sealed | id only - sealed | regular - sealed | rd only - sealed | rd only - unsealed |
| | unsealed | rd only - unsealed | id only - unsealed | regular - unsealed | rd only - sealed | rd only - unsealed |
| id only | code/uini | ***sverr*** | | | | |
| | sealed | id only - sealed | id only - sealed | regular - sealed | id only - sealed | id only - unsealed |
| | unsealed | id only - unsealed | id only - unsealed | regular - unsealed | id only - sealed | id only - unsealed |
| io | – | ***sverr*** | | | | |
| frame | uini | ***sverr*** | | | | |
| | unsealed | ***sverr*** | | | | |

Exception handling (interrupts, faults, traps)



*rte*



***sverr*** if
– not executed by the *core object*
– *x_exc_code_index* ≥ *x_exc_code_cxi*
– *x_exc_code_pnt*, *x_exc_code_xi*, *x_exc_code_cxi* do not describe a valid code object

**Causes for supervisor errors (list still incomplete)**

– *ccp*: argument is null or not a multiple of eight, invalid code attributes
– *alcb*: argument is null or not a multiple of eight
– *ciop*: device/δ too large or δ = 0
– *crop*, *cidp*, *rpr*, *seal*: inappropriate pointer or object
– physical memory access: misaligned address, access to pointer area
– *rte* executed in a code object other than the *core object*
– ...