# Overview

The primary extension to the RISCV ISA is the support for segmentation. The RISCV ISA does allow for both instruction and data base and bound registers. This is basic segmentation. However this segmentation is not available to machine level software. There are several desirable features missing. There aren’t enough segment registers. The addressable range of memory for the core can be extended by providing a segment shift rather than a simple addition for the base register. Modern operating systems also use additional segment registers to access thread local data and global data. There is also no stack bounds checking in RISCV.

The segmented core allows a larger address range to be used while retaining the small size of a 32 bit core. The address range is extended by 12 bits to 44 bits. A segmented system may provide a low overhead means of memory protection.

Base Instruction Formats

These are the basic RISCV instruction formats (for reference).

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 31 25 | | 24 20 | | 19 15 | 14 12 | 11 7 | | 6 0 |  |
| funct7 | | rs25 | | rs15 | funct3 | rd5 | | Opcode7 | R-Type |
| Imm11..0 | | | | rs15 | funct3 | rd5 | | Opcode7 | I-Type |
| Imm11..5 | | rs25 | | rs15 | funct3 | Imm4..0 | I0 | Opcode7 | S-Type |
| I12 | Imm10..5 | rs25 | | rs15 | funct3 | Imm4..1 | I11 | Opcode7 | SB-Type |
| Imm31..12 | | | | | | rd5 | | Opcode7 | U-Type |
| I20 | Imm10..5 | Imm4..1 | I11 | Imm19..15 | Imm14..12 | rd5 | | Opcode7 | UJ-Type |

# Greenfield Extensions

## Segment Registers

It is envisioned that user mode code will only use either the data segment or possibly the code segment.

The upper three bits of the virtual data address determine the segment register to use. The data segment is available for positive or negative offsets as shown in the mapping table below. This mapping allows short offsets (12 bit) to be used with most load and store instructions. Note that the upper bits of the program counter are not used to select the segment – there is no selection to be made, it is always the code segment. Note also that there is no stack segment, the stack should be contained in the data segment. Variables on the stack may be referenced with negative offsets while still retaining direct use of load and store instructions.

Because the upper three bits of the virtual address select the segment to use, the contiguous address range for data is limited to 29 bits (512MiB).

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| VA31..29 |  | Reset Value | CSR | Read /Write |  |
| 000 | DS | 00000h | 792h |  | data segment (positive offsets) |
| 001 | ES | 20000h | 793h |  |  |
| 010 | FS | 40000h | 794h |  |  |
| 011 | GS | 60000h | 795h |  |  |
| 100 | CS | 00000h | 791h | ro | code segment |
| 101 | CS |  |  |  | code segment |
| 110 | CS |  |  |  |  |
| 111 | DS |  | 792h |  | data segment (negative offsets) |
|  | ECS |  | 790h |  | exception code segment |
|  | DSL | FFFFFFFFh | 79Ah |  | data segment limit |
|  | ESL | FFFFFFFFh | 79Bh |  |  |
|  | FSL | FFFFFFFFh | 79Ch |  |  |
|  | GSL | FFFFFFFFh | 79Dh |  |  |
|  | CSL | FFFFFFFFh | 799h | ro | code segment limit |
|  | ECSL |  | 798h |  | exception code segment limit |
|  | SLL | 0 | 79Eh |  | stack lower limit |
|  | SLU | FFFFFFFFh | 79Fh |  | stack upper limit |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

The code segment is read-only from the CSR.

The exception code segment is loaded with the current code segment when an exception occurs. The ERET instruction will set both the program counter and code segment back to their prior values from the contents of the EPC and ECS registers.

On reset the code segment is set to zero and the program counter is set to $2000.

The stack lower and upper limit registers are checked against the stack pointer (x14) and frame pointer (x2) registers when a data access (load or store) occurs in the data segment.

## Additional Instructions

### Far Jump

A far jump instruction is used to set the code segment. The exception vector pointer (evec) may point to a far jump operation allowing control to pass to a different code segment. The far jump instruction is a 128 bit instruction. The far jump instruction is available only at the machine level. For user level code to access a different code segment a call to the operating system should be made with the ECALL instruction.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Limit32 | Segment32 | Offset32 | ACR16 | Opcode5 | nnnn | Opcode7 |
| Limit32 | Segment32 | Offset32 | 016 | 00000 | 0011 | 1111111 |

The ERET instruction may also be used to set the code segment and program counter.

# Brownfield Modifications

## Immediate Constants

If the X32 core parameter is set true then extended 32 bit constants may be used by the core. This is made possible by the use of a 128 bit instruction window into the cache (necessary for the far jump instruction).

For S and I type instruction formats (load, store and alu), if the most significant seven bits of the immediate constant are equal to 1000000b then the next 32 bit instruction word is used as the immediate constant. This saves a register and possibly an instruction over having to load a 32 bit constant into a register for use. Note that the extended constant doesn’t apply to SB (branch) and U type instructions. There isn’t a need for more than 13 bits of branch displacement.

For the CSR ‘I’ type instructions if the immediate value is equal to 10000b then the next 32 bit instruction word is used as the immediate value.

## Exceptions

When an exception occurs the code segment and limit are stored in the exceptioned code segment (ECS) and exceptioned code segment limit (ECSL) registers. This allows the ERET instruction to return to the point of the exception. Once the code segment is saved it is set to the value zero in order to access the exception vector address specified by the EVEC register. The exception table must be located in code segment zero (the first 4GiB of memory). A far jump may be made from the exception table to a handler in the far address space.

## MCAUSE Register

When a hardware interrupt occurs the most significant bit of the mcause register is set, the exception code is set to 15 ( a previously reserved value) and bits 4 through 12 of the register are set to a vector number supplied by the interrupt hardware.

|  |  |  |  |
| --- | --- | --- | --- |
| 31 | 30 13 | 12 4 | 3 0 |
| Interrupt | Unused | Vector Number | Exception Code |
| 11 | 018 | Vector9 | Fh4 |

## Instructions

### ERET

The ERET instruction also sets the code segment and code segment limit back to their originating values from the ECS and ECSL registers.

### Segment extract instructions

The segment extract instructions allow a segment value (base, upper limit, or lower limit) to be easily transferred to another register when given only the virtual address in a register.

### XSEG

XSEG extracts the segment base value given a virtual address in a register. The segment value extracted depends on the upper three bits of the register. The instruction is encoded as an I-type instruction under an unused SLLI funct3 code.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 60h7 | 05 | rs15 | 13 | rd5 | 13h7 | I-Type |

### XSEGU

XSEGU extracts the upper segment limit value given a virtual address in a register. The segment value extracted depends on the upper three bits of the register. The instruction is encoded as an I-type instruction under an unused SLLI funct3 code.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 61h7 | 05 | rs15 | 13 | rd5 | 13h7 | I-Type |

### XSEGL

XSEGL extracts the lower segment limit value given a virtual address in a register. The segment value extracted depends on the upper three bits of the register. Normally this value is zero except in the case where the register is the stack pointer or frame pointer and the address is in the data segment.

The instruction is encoded as an I-type instruction under an unused SLLI funct3 code.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 62h7 | 05 | rs15 | 13 | rd5 | 13h7 | I-Type |