Segmented Core

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 |

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. There are two unsegmented address ranges between $A0000000 and $DFFFFFFF. Note that a segmented address may also cover the range. 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 address range for data is limited to 29 bits (512MiB).

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| VA31..29 |  | Reset Value | CSR |  |
| 000 | DS | 00000h | 792h | data segment (positive offsets) |
| 001 | ES | 20000h | 793h |  |
| 010 | FS | 40000h | 794h |  |
| 011 | GS | 60000h | 795h |  |
| 100 | CS | 00000h | 791h | code segment |
| 101 | -- |  |  | unsegmented address range $Axxxxxxx,$Bxxxxxxx |
| 110 | -- |  |  | unsegmented address range $Cxxxxxxx,$Dxxxxxxx |
| 111 | DS |  | 792h | data segment (negative offsets) |
|  | ECS |  | 790h | exception code segment |

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.

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.

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

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

Brownfield Modifications

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, 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.