# DSD5

## Overview

DSD5 is a 64 bit RISC oriented core with a segmentation model applied. DSD5 borrows features from the RISC-V ISA although it is not a RISC-V core. There are relatively few different types of instructions and complex operations requiring multiple clock cycles like multiply and divide have been left out.

## Instruction Size

DSD5’s minimum instruction parcel size is a 32 bit word. Instructions may be more than one word in length.

# Control and Status Registers (CSR)

There could be up to four sets of CSR’s depending on the available core operating levels. The CSR set selected is chosen from the upper two bits of the CSR register number. Since the register number is a 13 bit field there could be up to 2048 CSR’s for each operating level.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Regno11 | Width |  |  |  |
| 0x000 | 64 |  | reserved – reads as zero |  |
| 0x00F | 64 | SP | stack pointer |  |
| 0x010 | 32 | DS | DS selector |  |
| 0x011 | 32 | ES |  |  |
| 0x012 | 32 | FS |  |  |
| 0x013 | 32 | GS |  |  |
| 0x014 | 32 | HS |  |  |
| 0x015 | 32 | JS |  |  |
| 0x016 | 32 | CS | CS is read-only |  |
| 0x017 | 32 | DS | same as register 0x10 |  |
| 0x018 | 32 | SS | stack selector |  |
| 0x019 | 32 | LDT | selector for local descriptor table |  |
| 0x01A | 32 | TR | task register – a write triggers a load of the task state segment if the TSS descriptor is valid. The TR may be set to zero. |  |
| 0x01B | 32 | TR | task register – a write only sets the task register, no load of task state occurs. Writing a zero is considered a bad TSS, |  |

# Segmentation

## Overview

Segmentation is a low overhead means of memory protection and virtualization. Providing separate protected address spaces for different applications is the job of the operating system. Ideally segmentation hardware should not be visible to the application. The application should appear as though it has a flat memory model. The core contains seven segment registers. The segmentation system is managed via a combination of hardware and software. Up to 256 privilege levels are available.

## Privilege levels

Memory access is available according to privilege levels. The segmentation system allows up to 256 privilege levels.

## Operating Levels

While the core supports 256 privilege levels there are four basic sets of privilege operating levels for the core. User, supervisor, hypervisor and machine levels are available. These four sets map onto the available privilege levels as the following table shows.

|  |  |  |
| --- | --- | --- |
| Privilege Level Set | Privilege Level |  |
| 0 | 0 | Machine mode privilege level |
| 1 | 1 | Hypervisor level |
| 2 | 2 to 7 | Supervisor levels |
| 3 | 8 to 255 | User levels |

## Segment Registers

There are seven segment registers in the architecture – CS,DS,ES,FS,GS,HS and JS. The stack is managed as part of the data segment and the stack bounds register is also loaded from a descriptor. Writing a selector value into a segment register causes the hidden portion of the register to be loaded from the descriptor table.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| VA63..61 |  | Reset Value | CSR | Read /Write |  |
| 000 | DS |  | 010h |  | data segment (positive offsets) |
| 001 | ES |  | 011h |  |  |
| 010 | FS |  | 012h |  |  |
| 011 | GS |  | 013h |  |  |
| 100 | HS |  | 014h |  |  |
| 101 | JS |  | 015h |  |  |
| 110 | CS | 00000h | 016h | ro | code segment |
| 111 | DS |  | 017h |  | data segment (negative offsets) |
|  | SS |  | 018h |  | stack bounds |
|  | LDT |  | 019h |  | local descriptor table selector |

## Selecting a segment register

A specific segment register for a memory operation may be selected using the upper three bits of the virtual data address. The upper three bits of the program counter do not select a segment since the segment association is always fixed to the code segment. Code addresses always use segment register #6 – the code segment.

## Selectors

The core uses selectors as a more compact way to represent segment registers. Rather than pass the entire segment descriptor to routines (256 bits) and have each routine check for privilege violations, the core uses 32 bit selectors. Privilege violations are checked for at the time the segment register components (base, limit and access rights) are loaded. The selector includes a field identifying the privilege level, and a second field identifying which segment descriptor the selector is associated with. The selector format is shown below.

#### Selector Format:

|  |  |  |
| --- | --- | --- |
| 31 24 | 23 | 22 0 |
| PL8 | T | Index23 |

PL8: the privilege level associated with the segment

Index23: the index into the descriptor table

T: 0 = global, 1 = local descriptor table

## Descriptors

### Memory Descriptors

Memory descriptors describe the location and size of memory segments. They have the following format:

|  |  |  |
| --- | --- | --- |
| n+3 | ~48 | ACR16 |
| n+2 | ~64 | |
| n+1 | Limit63..0 | |
| n | Base63..0 (Lower limit for stack) | |

#### The Access Rights Field (ACR16) – Memory Descriptor

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 15 |  |  | 12 | 11 | 10 | 9 | 8 | 7 0 |
| P | DC | ~ | 1/S | Ex | C/Stk | W/R | A | DPL8 |

P: 1 = segment present, 0 = segment not present

DC: 1 = descriptor cacheable, 0 = non-cacheable

S: 0 = system descriptor, 1 = memory descriptor

EX: 1 = executable, 0 = data

Code Segment Data Segment

C: 1= conforming Stk: 1=stack bounds

R: 1 = readable W: 1=writeable

A: 1= accessed

DPL8 = descriptor privilege level

#### Typical Values for ACR

9A00 – executable, readable code segment, privilege level zero

9200 – read/writeable data segment, privilege level zero

9600 – read / writeable stack segment, privilege level zero

### Stack Segment Descriptors

There is no base address specified for a stack segment descriptor instead a lower stack limit is specified in the descriptor. The stack must be part of the data segment and inherits the data segment base address. There isn’t a way to select the stack segment from the virtual address instead the stack bounds information is accessed as part of the data segment.

### System Segment Descriptors

System descriptors are identified by having bit12 of the access rights character set to zero. There are potentially sixteen different system descriptor types.

#### The Access Rights Field (ACR16) – System Descriptor

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 15 |  |  | 12 | 11 | 10 | 9 | 8 | 7 |  |  | 0 |
| P | DC | ~ | 0 | Type4 | | | | DPL8 | | | |

|  |  |  |
| --- | --- | --- |
| Type4 | Gate |  |
| 0 | unused |  |
| 1 | TSS descriptor |  |
| 2 | LDT descriptor |  |
| 3 | Busy TSS descriptor |  |
| 4 | Call gate |  |
| 5 | Task Gate |  |
| 6 | Interrupt Gate |  |
| 7 | Trap gate |  |

#### Compressed Descriptors

A compressed version of descriptors is used by the exception descriptor table. Since this table can only hold interrupt, trap or task gates and all fields of a descriptor are not required the descriptors are stored in a compressed format. Descriptor entries in the EDT require only two words of information.

#### Interrupt Gate Descriptor

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 86h8 | DPL8 | ~11 | 05 | Selector31..0 |
| Offset63..0 | | | | |

#### Trap Gate Descriptor

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 87h8 | DPL8 | ~11 | 05 | Selector31..0 |
| Offset63..0 | | | | |

#### Task Gate Descriptor

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 85h8 | DPL8 | ~11 | 05 | Selector31..0 |
| 063..0 | | | | |

#### Call Gate Descriptor

|  |  |  |  |
| --- | --- | --- | --- |
| ACR16 | ~11 | N5 | Selector31..0 |
| Offset63..0 | | | |

#### Task Gates

The task exit gate points to code responsible for performing task exit chores. The task exit gate automatically stores the following registers in the current tasks task state segment: cs:pc, mws, sp0, sp1, sp2, sp3, ds0, ds1, ds2, ds3, ss0, ss1, ss2, ss3, r1, r2, and r3. The state stored automatically is just enough to make it easier to implement the remainder of the task switching code. Rather than performing a full task switch automatically which would save all the registers then restore all the registers for the new task, only a partial save is done. The benefit of doing this is that it holds up the core for far fewer clock cycles. It takes only about 20 clock cycles to perform the partial save. It could take on the order of 200 clock cycles to load all the registers including performing segment register loads. The partial save offers a much lower latency.

#### Task State Segment (TSS) Descriptors

TSS descriptors point to a block of memory that contains the core’s state for task management.

## The Task State Segment

Loading a selector value into the task register (TR) causes the state contained in the specified TSS to be loaded into the core. The task register is loaded automatically in response to a task switch caused by a task gate. The core’s current state may be saved in the TSS using the STSS instruction.

The task state segment has the following format:

|  |  |  |
| --- | --- | --- |
| n+344 | ~32 | ldt |
| n+336 | cs | js |
| n+328 | hs | gs |
| n+320 | fs | es |
| n+312 | ss[3] | ss[2] |
| n+304 | ss[1] | ss[0] |
| n+296 | ds[3] | ds[2] |
| n+288 | ds[1] | ds[0] |
| n+280 | r30 | |
| n+272 | r29 | |
| n+264 | r28 | |
| n+256 | r27 | |
| n+248 | r26 | |
| n+240 | r25 | |
| n+232 | r24 | |
| n+224 | r23 | |
| n+216 | r22 | |
| n+208 | r21 | |
| n+200 | r20 | |
| n+192 | r19 | |
| n+184 | r18 | |
| n+176 | r17 | |
| n+168 | r16 | |
| n+160 | r15 | |
| n+152 | r14 | |
| n+144 | r13 | |
| n+136 | r12 | |
| n+128 | r11 | |
| n+120 | r10 | |
| n+112 | r9 | |
| n+104 | r8 | |
| n+96 | r7 | |
| n+88 | r6 | |
| n+80 | r5 | |
| n+72 | r4 | |
| n+64 | r3 | |
| n+56 | r2 | |
| n+48 | r1 | |
| n+40 | SP[3] | |
| n+32 | SP[2] | |
| n+24 | SP[1] | |
| n+16 | SP[0] | |
| n+8 | ~32 | MSW |
| n | PC | |

# Exceptions

## Exception Descriptor Table

The exception descriptor table contains compressed descriptors that point to exception handling routines. There may only be interrupt, trap, or task gate descriptors in the exception descriptor table. Interrupt gates and trap gates are identical except that interrupt gates mask off further interrupts while trap gates do not.

The exception vector table provides room for up to 512 compressed descriptors.

### Exception Vector Table Usage

The following table outlines which vector is used for a given purpose. These vectors are specific to DSD5. Under the HW column an ‘x’ indicates that the interrupt is internally generated by the processor; the vector is hard-wired to that use. An ‘e’ indicates an externally generated interrupt, the usage may vary depending on the system.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Vecno |  | HW | Description |  |
| 0 |  |  |  |  |
| 1 |  |  |  |  |
| 2 |  |  | FMTK Scheduler |  |
| 449 | KRST | e | Keyboard reset interrupt |  |
| 450 | MSI | e | Millisecond Interrupt |  |
| 451 | TICK | e | FMTK Tick Interrupt |  |
| 463 | KBD | e | Keyboard interrupt |  |
| 488 | DBZ | x | divide by zero |  |
| 489 | OFL | x | overflow |  |
| 493 | FLT | x | floating point exception |  |
| 497 | EXF | x | Executable fault |  |
| 498 | DWF | x | Data write fault |  |
| 499 | DRF | x | data read fault |  |
| 500 | SBV | x | segment bounds violation |  |
| 501 | PRIV | x | privilege level violation |  |
| 502 | STV | x | segment type violation |  |
| 503 | SNP | x | segment not present |  |
| 504 | STF | x | stack fault |  |
| 505 | CPF | x | code page fault |  |
| 506 | DPF | x | data page fault |  |
| 508 | DBE | x | data bus error |  |
|  |  |  |  |  |
| 510 | NMI | x | Non-maskable interrupt |  |
|  |  |  |  |  |

# Memory Operations:

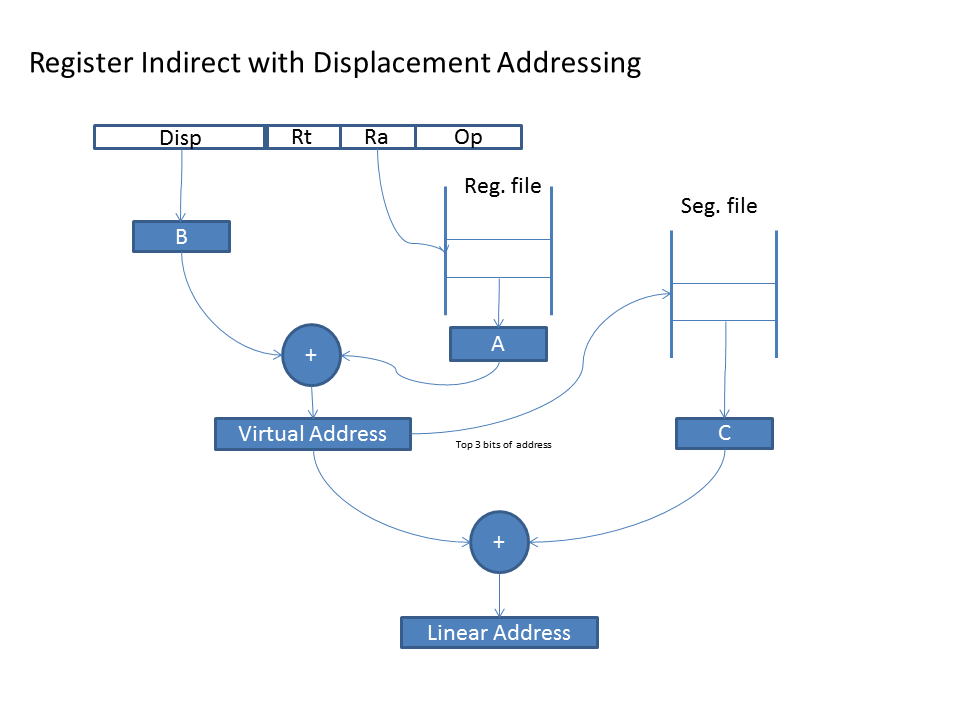
## Basic Operations

Basic memory operations include loads, stores, and subroutine calls. Other than those operations there are no other instructions that access memory.

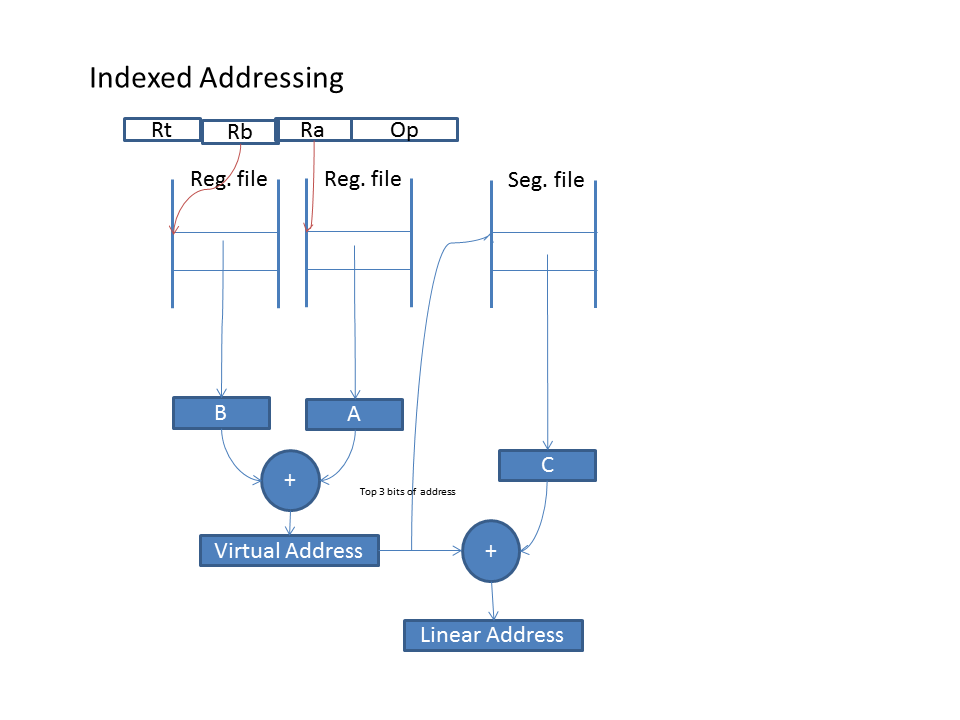
## Memory Addressing Modes

The core supports both register indirect with displacement and indexed addressing. Indexed addressing is supported only with the general purpose register load store operations.

Register indirect addressing looks up both a register value and a segment register value from the register files and adds a displacement from the instruction to form an address.



Indexed addressing looks up two register file values and a segment register value then adds all three values to form an address.



# Detailed Instruction Set

## ADDI

Description:

Calculate the sum of a register and an immediate value and place the result in the target register.

Instruction Format:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| w0 | Immediate15 | | Rt5 | | Ra5 | 04h7 | | I15 |
| w0 | 80h8 | ~7 | Rt5 | Ra5 | | | 04h7 | I32 |
| w1 | Immediate31..0 | | | | | | |
| w0 | 81h8 | ~7 | Rt5 | Ra5 | | | 04h7 | I64 |
| w1 | Immediate31..0 | | | | | | |
| w2 | Immediate63..32 | | | | | | |

Operation:

Rt = Ra + Imm

Clock Cycles: 1

## ANDI

Description:

Perform the bitwise ‘AND’ of a register and an immediate value and place the result in the target register.

Instruction Format:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| w0 | Immediate15 | | Rt5 | | Ra5 | 08h7 | | I15 |
| w0 | 80h8 | ~7 | Rt5 | Ra5 | | | 08h7 | I32 |
| w1 | Immediate31..0 | | | | | | |
| w0 | 81h8 | ~7 | Rt5 | Ra5 | | | 08h7 | I64 |
| w1 | Immediate31..0 | | | | | | |
| w2 | Immediate63..32 | | | | | | |

Operation:

Rt = Ra & Imm

Clock Cycles: 1

## BEQ

Description:

Branch if two registers specified are equal. The 15 bit displacement is shifted left and sign extended before being added to the program counter. The branch range is then +/- 8k words.

Instruction Format:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Displacement15..1 | Rb5 | Ra5 | 13 | 04 |

Operation:

if (Ra = Rb)

pc <= pc + displacement \* 2

Clock Cycles: 3 if branch is taken, otherwise 1

## BNE

Description:

Branch if two registers specified are not equal. The 15 bit displacement is shifted left and sign extended before being added to the program counter. The branch range is then +/- 8k words.

Instruction Format:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Displacement15..1 | Rb5 | Ra5 | 13 | 14 |

Operation:

if (Ra <> Rb)

pc <= pc + displacement \* 2

Clock Cycles: 3 if branch is taken, otherwise 1

## BRK – Break

Description:

Execute a breakpoint interrupt. The interrupt executed is identified by a nine bit vector. The vector is an index into the exception descriptor table. This instruction may be followed by up to two instruction words of information.

Instruction Format:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| w0 | ~15 | | ~ | Vector9 | 00h7 | |  |
| w0 | 80h8 | ~7 | ~ | Vector9 | | 00h7 | I32 |
| w1 | Data31..0 | | | | | |
| w0 | 81h8 | ~7 | ~ | Vector9 | | 00h7 | I64 |
| w1 | Data31..0 | | | | | |
| w2 | Data63..32 | | | | | |

Operation:

Clock Cycles: 10

## LDT

Description:

Loads into either the exception or the global descriptor table pointer register and limit register. The address specified is a non-segmented 76 bit physical address. The address is calculated as the sum of the 76 bit immediate constant and the contents of register Ra. The address should be word aligned. This instruction is only available at machine level. The global descriptor table register must be loaded before segment registers can be loaded. The exception descriptor table register must be loaded before interrupts and exceptions can be executed.

Instruction Format:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Immediate19..3 | | Rt3 | Ra5 | h7 |
| Immediate51..20 | | | | |
| ~8 | Immediate75..52 | | | |

|  |  |
| --- | --- |
| Rt3 | Register Loaded |
| 1 | EDT |
| 2 | GDT |
|  |  |

Format of data loaded:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Base address63..4 | | | | 04 |
| ~4 | Table Limit27..0 | ~20 | Base75..64 | |

The base address should be 16 byte aligned.

The table limit field accommodates the maximum number of entries that can be specified in a selector. In practice this limit will be set to a lower value.

The maximum number of entries in the exception descriptor table is 512.

## STSS

Description:

Store registers to the task state segment. Which registers are stored is controlled by a bit mask immediate constant supplied in the instruction. The task state segment is located via the task register which must have been previously set. The value of the program counter stored is the address of the STSS instruction plus 12. This leaves room for a branch instruction to be inserted in case the TSS including the PC is loaded.

Segment registers are only stored in pairs to increase memory efficiency.

This instruction is not available in user mode.

Instruction Format:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| w0 | 81h8 | ~7 | 05 | 05 | 55h7 | I64 |
| w1 | Immediate31..0 | | | | |
| w2 | Immediate63..32 | | | | |
| w3 | <reserved for branch instruction> | | | | |  |

Registers Stored According to Bit (1 = store, 0=skip)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Bit | Register | Bit | Register | Bit | Register |
| 0 | PC | 16 | r11 | 32 | r27 |
| 1 | MSW | 17 | r12 | 33 | r28 |
| 2 | sp[0] | 18 | r13 | 34 | r29 |
| 3 | sp[1] | 19 | r14 | 35 | r30 |
| 4 | sp[2] | 20 | r15 | 36 | ds[1],ds[0] |
| 5 | sp[3] | 21 | r16 | 37 | ds[3],ds[2] |
| 6 | r1 | 22 | r17 | 38 | ss[1],ss[0] |
| 7 | r2 | 23 | r18 | 39 | ss[3],ss[2] |
| 8 | r3 | 24 | r19 | 40 | fs,es |
| 9 | r4 | 25 | r20 | 41 | hs,gs |
| 10 | r5 | 26 | r21 | 42 | cs,js |
| 11 | r6 | 27 | r22 | 43 | ldt |
| 12 | r7 | 28 | r23 |  |  |
| 13 | r8 | 29 | r24 |  |  |
| 14 | r9 | 30 | r25 |  |  |
| 15 | r10 | 31 | r26 |  |  |

# Opcode Formats:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 32 | Immediate15 | | | | | | | Rt5 | | | | | | Ra5 | | Opcode7 | | | | I15 |
| 64 | 80h8 | | | | ~7 | | | Rt5 | | | | | Ra5 | | | | Opcode7 | | | I32 |
| Immediate31..0 | | | | | | | | | | | | | | | | | | |
| 96 | 81h8 | | | | ~7 | | | Rt5 | | | | | Ra5 | | | | Opcode7 | | | I64 |
| Immediate31..0 | | | | | | | | | | | | | | | | | | |
| Immediate63..32 | | | | | | | | | | | | | | | | | | |
| 32 | Funct7 | | | ~3 | | Rt5 | | Rb5 | | | | | | Ra5 | | | Opcode7 | | | R2 |
|  | Funct5 | Rt5 | | | | Rc5 | | Rb5 | | | | | | Ra5 | | | Opcode7 | | | R3 |
| 32 | Displacement15..1 | | | | | | | Imm4..0 | | | | | | Ra5 | | | 23 | Cnd4 | | Bri5 |
| 64 | Displacement15..1 | | | | | | | 10h5 | | | | | | Ra5 | | | 23 | Cnd4 | | Bri32 |
| Immediate31..0 | | | | | | | | | | | | | | | | | | |
|  | Displacement15..1 | | | | | | | 11h5 | | | | | | Ra5 | | | 23 | Cnd4 | | Bri64 |
|  | Immediate31..0 | | | | | | | | | | | | | | | | | | |
|  | Immediate63..32 | | | | | | | | | | | | | | | | | | |
| 32 | Displacement15..1 | | | | | | | Rb5 | | | | Ra5 | | | | 13 | | Cnd4 | | Brr |
| 32 | Address25..1 | | | | | | | | | | | | | | | Opcode7 | | | | J26 |
| 64 | 80h8 | | | | ~17 | | | | | | | | | | | Opcode7 | | | | J32 |
| Address31..1 | | | | | | | | | | | | | | | | | | ~1 |
| 96 | 81h8 | | | | ~17 | | | | | | | | | | | Opcode7 | | | | J64 |
| Address31..1 | | | | | | | | | | | | | | | | | | ~1 |  |
| Address63..32 | | | | | | | | | | | | | | | | | | |  |
| 128 | 81h8 | | | | ~17 | | | | | | | | | | | Opcode7 | | | | JF |
| Address31..1 | | | | | | | | | | | | | | | | | | ~1 |
| Address63..32 | | | | | | | | | | | | | | | | | | |
| Selector31..0 | | | | | | | | | | | | | | | | | | |
|  | CSR13 | | | | | | Op2 | | Rt5 | | | Ra5 | | | | Opcode7 | | | | CSR |
|  | CSR13 | | | | | | Op2 | | Rt5 | | | Imm4..0 | | | | Opcode7 | | | | CSRI |
|  |  | |  | | | |  | |  | | |  | | | |  | | | |  |
| 128 | Address23..0 | | | | | | | | | | | | | | | Opcode8 | | | | LDI seg,# |
| Address55..24 | | | | | | | | | | | | | | | | | | |
| ACR16 | | | | | | | | | ~5 | Seg3 | | | | Address63..56 | | | | |
| Selector31..0 | | | | | | | | | | | | | | | | | | |
|  |  | |  | | | |  | |  | | |  | | | |  | | | |  |

|  |  |  |
| --- | --- | --- |
| Op2 | CSR | CSRI |
| 0 | CSRRW | CSRRWI |
| 1 | CSRRS | CSRRSI |
| 2 | CSRRC | CSRRCI |
|  |  |  |

# Opcode Maps

## Major Opcodes

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | x0 | x1 | x2 | x3 | x4 | x5 | x6 | x7 | x8 | x9 | xA | xB | xC | xD | xE | xF |
| 0x | BRK | RTS |  |  | ADDI | CMPI | CMPUI |  | ANDI | ORI | EORI |  |  |  |  |  |
| 1x | BEQ | BNE | BLT | BGE | BLE | BGT | BLTU | BGEU | BLEU | BGTU |  |  |  |  |  | CSR |
| 2x | BEQI | BNEI | BLTI | BGEI | BLEI | BGTI | BLTUI | BGEUI | BLEUI | BGTUI |  |  |  |  |  | CSRI |
| 3x | {r2} | {r3} |  |  | JMP | JSR |  |  | LDT |  |  |  |  |  |  |  |
| 4x | LB | LBU | LC | LCU | LH | LHU | LW | LWR |  |  |  |  |  |  | EXB | EXL |
| 5x | SB | SC | SH | SW | SWC | STSS |  |  |  |  |  |  |  |  |  |  |
| 6x |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 7x | NOP |  | RTE | RTF | JMF | JSF |  | SYNC | MEMSB | MEMDB | CLI | SEI |  |  |  |  |

## {r2} Funct7 Opcodes

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | x0 | x1 | x2 | x3 | x4 | x5 | x6 | x7 | x8 | x9 | xA | xB | xC | xD | xE | xF |
| 0x |  |  |  |  | ADD | CMP | CMPU | SUB | AND | OR | EOR |  | NAND | NOR | ENOR |  |
| 1x |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 2x |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 3x |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 4x | LBX | LBUX | LCX | LCUX | LHX | LHUX | LWX | LWRX |  |  |  |  |  |  |  |  |
| 5x | SBX | SCX | SHX | SWX | SWCX |  |  |  |  |  |  |  |  |  |  |  |
| 6x |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 7x | NOP |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

## {r3} Funct5 Opcodes

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | x0 | x1 | x2 | x3 | x4 | x5 | x6 | x7 | x8 | x9 | xA | xB | xC | xD | xE | xF |
| 0x | CAO | CSO | CAC | CSC |  |  |  |  | CAOI | CSOI | CACI | CSCI |  |  |  |  |
| 1x |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |