## Operating Modes

The core operates in one of four modes, two user, a supervisor or system mode and a debug mode. There is a complete set of registers for each operating mode including segment and flags registers. Some instructions are not available in user mode and will cause a privilege violation if an attempt to execute them is made. System mode is automatically switched to when the BRK instruction is executed. Debug mode may be entered using the DBG instruction.

|  |  |
| --- | --- |
| Mode |  |
| 0 | user 1 |
| 1 | user 2 |
| 2 | system |
| 3 | debug |

# Programming Model

## General Purpose Registers

There are four complete sets of registers. One for each operating mode. Register r0 always reads as zero. Register r31 references the stack pointer.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| User1 |  | User2 |  | System |  | Debug |  |
| r0 |  | r0 |  | r0 |  | r0 |  |
| r1 |  | r1 |  | r1 |  | r1 |  |
| r2 |  | r2 |  | r2 |  | r2 |  |
| r3 |  | r3 |  | r3 |  | r3 |  |
| r4 |  | r4 |  | r4 |  | r4 |  |
| r5 |  | r5 |  | r5 |  | r5 |  |
| r6 |  | r6 |  | r6 |  | r6 |  |
| r7 |  | r7 |  | r7 |  | r7 |  |
| r8 |  | r8 |  | r8 |  | r8 |  |
| r9 |  | r9 |  | r9 |  | r9 |  |
| r10 |  | r10 |  | r10 |  | r10 |  |
| r11 |  | r11 |  | r11 |  | r11 |  |
| r12 |  | r12 |  | r12 |  | r12 |  |
| r13 |  | r13 |  | r13 |  | r13 |  |
| r14 |  | r14 |  | r14 |  | r14 |  |
| r15 |  | r15 |  | r15 |  | r15 |  |
| r16 |  | r16 |  | r16 |  | r16 |  |
| r17 |  | r17 |  | r17 |  | r17 |  |
| r18 |  | r18 |  | r18 |  | r18 |  |
| r19 |  | r19 |  | r19 |  | r19 |  |
| r20 |  | r20 |  | r20 |  | r20 |  |
| r21 |  | r21 |  | r21 |  | r21 |  |
| r22 |  | r22 |  | r22 |  | r22 |  |
| r23 |  | r23 |  | r23 |  | r23 |  |
| r24 |  | r24 |  | r24 |  | r24 |  |
| r25 |  | r25 |  | r25 |  | r25 |  |
| r26 |  | r26 |  | r26 |  | r26 |  |
| r27 |  | r27 |  | r27 |  | r27 |  |
| r28 |  | r28 |  | r28 |  | r28 |  |
| r29 |  | r29 |  | r29 |  | r29 |  |
| r30 / BP |  | r30 /BP |  | r30 / BP |  | r30 / BP |  |
| usp1 |  | usp2 |  | ssp |  | dsp |  |
| Segment Registers | | | | | | |
| DS |  | DS |  | DS |  | DS |  |
| ES |  | ES |  | ES |  | ES |  |
| FS |  | FS |  | FS |  | FS |  |
| GS |  | GS |  | GS |  | GS |  |
| CS |  | CS |  | CS |  | CS |  |
| Condition Codes | | | | | | |
| CC |  | CC |  | CC |  | CC |  |

## Special Registers

### Machine Status Word (MSW)

The machine status word has the following format:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 31 10 | 9 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| ~22 | Mode | NF | VF | ~ | RF | OF | IM | ZF | CF |

There are separate carry, zero, negative, odd, overflow flags for each operating mode. Only the flags relevant to the current operating mode are visible in the MSW.

There is only a single reservation flag (RF), interrupt mask (IM) and mode indicator.

On reset the interrupt mask flag is set and the mode is set to mode #2.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 31 10 | 9 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| ~22 | 2 | X | X | ~ | X | X | 1 | X | X |

### Vector Table Code Segment

The vector table code segment register locates the vector table in memory and is automatically loaded into the code segment when an interrupt occurs. The offset portion of the address is determined by the vector number. On reset the VCS register is set to $FFFFFFFF placing the vector table in the highest 4kW of memory.

|  |
| --- |
| 31 0 |
| Segment31..0 |

# Interrupts and Exceptions

## Reset

On reset system mode is selected with interrupts masked. The VCS register is loaded with $FFFFFFFF placing the interrupt vector table in the highest 4kW of the memory space. A BRK instruction is executed for the reset vector. Since the BRK instruction writes to the stack, the data segment for mode #2 is also setup with DS = $00000000 and DSL = $FFFFFFFF. The system stack pointer is initialized to $100. Stack bounds are set to $0000 and $0100 for lower and upper bounds.

## Hardware Interrupts

Hardware interrupts are implemented with the BRK instruction which is automatically forced into the instruction stream when an interrupt occurs. The vector portion of the BRK instruction comes from an external vector bus, allowing the interrupt controller to select any vector in the vector table.

## Vector Table

When an interrupt occurs the processor is vectored to a location in the vector table. The program counter and code segment are set to point to the vector entry. The program counter value comes from the corresponding break (BRK) instruction. The code segment value is set to the vector table code segment (VCS) register. The code segment limit is set to the size of the vector table (2047). A subsequent far jump will reset the program counter, code segment and code limit.

The vector table is located in memory the vector table code segment register (VCS) which contains the segment value for the table. At reset the vector table is located at $FFFFFFFF:0000000. This places the vector table at physical address $FFFFFFFF000 – the last 4kW area of memory.

There are 512 entries in the vector table. Each entry in the vector table is four words in size. This is just enough room to allow a far jump instruction to be coded in the vector.

# Segmentation

The core features a segmented address space as a simple means of memory management and protection. If an attempt is made to access memory outside of the bounds set in one of the limit registers, then a software exception occurs. Segmentation is supported with four data segments and a code segment. There is no stack segment; the stack is part of the data segment. However the stack has its own bounds registers. The segment registers are accessible only in a supervisory mode.

## Address Formation

A segmented address is formed by adding a segment register shifted to the left twelve times to the virtual address formed by the instruction.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  | |  |  | |
|  |  |  | |  | Register29 | |
|  |  |  | |  | + | |
|  |  |  | |  | Offset29 | |
|  |  |  | |  | + | |
|  |  |  | Segment32 | | | 000h12 |
|  |  |  | = | | | |
|  |  |  | Linear Address44 | | | |

There are five segment registers available in the supervisor mode programming model.

The upper three bits of an effective address identify the segment register to be used in address formation. The following table shows how the upper bits map to segment registers.

|  |  |  |
| --- | --- | --- |
| EA31..29 |  | Segment register to use |
| 000 | DS | data segment |
| 001 | ES |  |
| 010 | FS |  |
| 011 | GS |  |
| 100 | CS | code segment |
| 101 | --- | reserved |
| 110 | --- | reserved |
| 111 | DS | data segment |

Note that the DS register is listed twice. The values 000 and 111 are carefully chosen for the DS register as they allow only 16 bit constants to be used in forming the effective address. This is primarily for use with variables located on the stack which may have negative offsets as well as positive ones.

Note also there is no stack segment. The stack is located in the data segment. In a user application there should be only either the data segment (code 000, 111) or code segment (100) in use. The user application does not need to have knowledge of segmentation.

## Bounds Checking

### Program Counter

Bounds checking of the program counter is only performed at instruction fetch time for the first word of the instruction fetched. Since an instruction fetch may result in the fetch of more than one word the program counter may exceed the bound limit during the fetch process. The longest instruction is three words in length. The segment limit should be set to accommodate the case where up to two words may be fetched beyond the limit.

### Stack

While the stack is located in the data segment, it has it’s own pair of bounds limiting registers. There is both a lower and upper bound to the stack. If the stack pointer falls below the lower bounds plus the stack margin a stack overflow results and a stack fault exception occurs. This may allow the size of the stack to be extended downwards. The margin is present to allow information to be stored on the stack during exception processing. The margin is set to four words by default and is a core parameter. For load and store operations if registers 30 or 31 are used then stack bounds are checked instead of data bounds.

### Data

Bounds checking against one of the data segment limits is performed before a load or store operation takes place and registers 0 to 29 are in use. If the virtual address exceeds the bounds then a bounds exception will occur.

# Instruction Encoding

## Immediate Constants

If bits 31 to 26 of the instruction word for an immediate operate instruction are set to 20h then the next instruction word is fetched from memory and used as the constant. Otherwise the 17 bit immediate constant encoded in the instruction is sign extended.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 31 26 | 25 15 | 14 10 | 9 5 | 4 0 |
| 100000b | X11 | Rt5 | Ra5 | Opcode5 |
| Immediate32 | | | | |

# Detailed Instruction Set Summary

### ADD – Register Register ADD

**Description:**

The sum of two registers is placed into a target register.

**Instruction Format:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 31 27 | 26 20 | 19 15 | 14 10 | 9 5 | 4 0 |
| 04h5 | ~7 | Rt5 | Rb5 | Ra5 | 02h5 |

**Clock Cycles:** 3

**Operation:**

**Rt = Ra + Rb**

**Exceptions:** none

### ADDI – Register Immediate ADD

**Description:**

The sum of a register and an immediate value is placed into a target register. The constant value is sign extended before use.

**Instruction Format:**

|  |  |  |  |
| --- | --- | --- | --- |
| 31 15 | 14 10 | 9 5 | 4 0 |
| Immediate16..0 | Rt5 | Ra5 | 04h5 |

**Clock Cycles:** 3

**Operation:**

**Rt = Ra + Rb**

**Exceptions:** none

### AND – Register Register Bitwise AND

**Description:**

The bitwise ‘And’ of two registers is placed into a target register.

**Instruction Format:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 31 27 | 26 20 | 19 15 | 14 10 | 9 5 | 4 0 |
| 08h5 | ~7 | Rt5 | Rb5 | Ra5 | 02h5 |

**Clock Cycles:** 3

**Operation:**

**Rt = Ra & Rb**

**Exceptions:** none

### ANDI – Register Immediate Bitwise AND

**Description:**

The bitwise ‘AND’ of a register and an immediate value is placed into a target register. The constant value is sign extended before use.

**Instruction Format:**

|  |  |  |  |
| --- | --- | --- | --- |
| 31 15 | 14 10 | 9 5 | 4 0 |
| Immediate16..0 | Rt5 | Ra5 | 08h5 |

**Clock Cycles:** 3

**Operation:**

**Rt = Ra & Rb**

**Exceptions:** none

### Bcc – Conditional Branch

**Description:**

Changes the program flow based on a branch condition. The branch condition is tested against the flags in the machine status word. The sign extended immediate value is added to the program counter if the branch is taken, otherwise the program counter is unchanged.

**Instruction Format:**

|  |  |  |  |
| --- | --- | --- | --- |
| 31 15 | 14 10 | 9 5 | 4 0 |
| Immediate16..0 | Cond5 | Ra5 | 03h5 |

**Clock Cycles:** 2 or 3 if taken

**Operation:**

|  |  |  |  |
| --- | --- | --- | --- |
| Cond5 | Mnemonic |  | Flag Test |
| 00 | BRA | always branch | 1 |
| 01 | BSR | branch to subroutine | 1 |
| 02 | BHI | branch if higher | !cf & !zf |
| 03 | BLS | branch if lower or same | cf | zf |
| 04 | BHS | branch if higher or same | !cf |
| 05 | BLO | branch if lower | cf |
| 06 | BNE | not equal | !zf |
| 07 | BEQ | equal | zf |
| 08 | BVC | overflow clear | !vf |
| 09 | BVS | overflow set | vf |
| 0A | BPL | plus (>=0) | !nf |
| 0B | BMI | minus (<0) | nf |
| 0C | BGE | greater or equal | (nf & vf)|(!nf & !vf) |
| 0D | BLT | less than | (nf & !vf)|(!nf & vf) |
| 0E | BGT | greater than | (nf & vf & !zf)|(!nf & !vf & zf) |
| 0F | BLE | less than or equal | zf | (nf & !vf) | (!nf & vf) |
| 10 | BEV | even | !of |
| 11 | BOD | odd | of |
| 12 | BPO | positive (>0) | !nf & !zf |
| 13 |  |  |  |
| 14 |  |  |  |
| 15 |  |  |  |
| 16 |  |  |  |
| 17 |  |  |  |
| 18 | BRZ | register is zero | Ra==0 |
| 19 | BNZ | register is not zero | Ra!=0 |
| 1A | BRP | register is plus | !Ra[31] |
| 1B | BRN | register is negative | Ra[31] |
| 1C | BXC | external signal clear | !xf |
| 1D | BXS | external signal set | xf |
| 1E | BRC | reservation clear | !rf |
| 1F | BRS | reservation set | rf |

### BRK – Software break

**Description:**

The code segment limit, code segment, program counter and machine status word are all stored on the supervisor mode stack. Program flow is then transferred to an entry in the vector table. The vector table entry may then perform a far jump to interrupt handling code. The vector table entry to vector to is identified by a nine bit field in the BRK instruction, this allows up to 512 interrupt vectors.

The vector table is located by the vector code segment register (VCS) which should point to the memory page of the start of the vector table.

**Instruction Format:**

|  |  |  |
| --- | --- | --- |
| 31 14 | 13 5 | 4 0 |
| ~18 | Vector9 | 00h5 |

**Clock Cycles:** 7 (2+5 memory)

**Operation:**

**memory[--ssp] = csl**

**memory[--ssp] = cs**

**memory[--ssp] = pc**

**memory[--ssp] = msw**

**csl = 2047**

**cs = vcs**

**pc = vector \* 4**

**Exceptions:** DBE, DBG, TLB

### CMP - Register-Immediate Compare

**Description:**

The register immediate compare instruction compares a register to an immediate value and sets the flags in the status register as a result. Both a signed and unsigned comparison take place at the same time.

**Instruction Format:**

|  |  |  |  |
| --- | --- | --- | --- |
| 31 15 | 14 10 | 9 5 | 4 0 |
| Immediate16..0 | ~5 | Ra5 | 06h5 |

**Clock Cycles:** 3

**Operation:**

### JMF – Far Jump to new segment

**Description:**

This instruction performs a jump to a far address setting a new program counter, code segment and code segment limit values. This instruction is allowed only in supervisor mode. It is typically used in the vector table to locate an interrupt routine. It may also be used at task startup to set the starting address.

**Instruction Format:**

|  |  |  |  |
| --- | --- | --- | --- |
| 31 5 | | | 4 0 |
| Offset26..0 | | | 14h5 |
| Segment31..0 | | | |
| Ts3 | ~2 | Segment Limit26..0 | |

**Clock Cycles:** 3

**Operation:**

**Exceptions:** DBE, DBG, TLB, PRIV

### JMP – Jump within segment

**Description:**

This instruction performs a jump to a new address setting a new program counter value.

**Instruction Format:**

|  |  |
| --- | --- |
| 31 5 | 4 0 |
| Offset26..0 | 10h5 |

**Clock Cycles:** 3

**Operation:**

### Exceptions: DBE, DBG, TLB

### MOV – Move Registers Between Sets

**Description:**

A register is moved from a source set to a destination set.

**Instruction Format:**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 31 27 | 26 23 | 22 20 | 19 15 | 14 13 | 12 10 | 9 5 | 4 0 |
| 03h5 | ~4 | Ts3 | Rt5 | ~2 | Ss3 | Ra5 | 02h5 |

**Clock Cycles:** 3

**Operation:**

**Exceptions:** DBE, DBG, TLB, MOV

|  |  |  |  |
| --- | --- | --- | --- |
| Ts3/Ss3 |  | Register Set |  |
| 0 | usr | user registers |  |
| 1 | usr | user registers |  |
| 2 | sys | supervisor registers |  |
| 3 | dbg | debug registers |  |
| 7 |  | current register set |  |

Example

Return a value from a system service routine to user mode.

mov usr:r1,r1

Send parameters to system routine from user mode

mov sys:r1,r1

mov sys:r2,r2

mov sys:r3,r3

mov sys:r4,r4

### MFSPR – Move From Special Purpose Register

**Description:**

The target register is loaded with the value from the special purpose register. This instruction is limited to execution is supervisor mode otherwise a privilege fault will occur.

**Instruction Format:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 31 27 | 26 20 | 19 15 | 14 10 | 9 5 | 4 0 |
| 19h5 | Spr7 | Rt5 | ~5 | ~5 | 02h5 |

**Clock Cycles:** 3

**Operation:**

**Exceptions:** DBE, DBG, TLB, PRIV

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Spr7 | r = read only |  | Description |  |
| 00 |  |  | reserved |  |
| 01 |  |  |  |  |
| 02 |  | VCS | vector table code segment |  |
| 03 |  | MSW | machine status word |  |
| 04 |  |  |  |  |
| 05 |  |  |  |  |
| 08 |  | DS[0] | user data segment |  |
| 09 |  | DS[1] | user data segment |  |
| 0A |  | DS[2] | supervisor data segment |  |
| 0B |  | DS[3] | debug data segment |  |
| 0C |  | DSL[0] | user data segment limit |  |
| 0D |  | DSL[1] | user data segment limit |  |
| 0E |  | DSL[2] | supervisor data segment limit |  |
| 0F |  | DSL[3] | debug data segment limit |  |
| 10 |  | SL[0] | user stack lower bound |  |
| 11 |  | SL[1] | user stack lower bound |  |
| 12 |  | SL[2] | supervisor stack lower bound |  |
| 13 |  | SL[3] | debug stack lower bound |  |
| 14 |  | SU[0] | user stack upper bound |  |
| 15 |  | SU[1] | user stack upper bound |  |
| 16 |  | SU[2] | supervisor stack upper bound |  |
| 17 |  | SU[3] | debug stack upper bound |  |
| 18 |  | usp1 | user stack pointer |  |
| 19 |  | usp2 | user stack pointer |  |
| 1A |  | ssp | system stack pointer |  |
| 1B |  | dsp | debug stack pointer |  |
| 20 | r | cs[0] | code segment for user mode |  |
| 21 | r | cs[1] | code segment for user mode |  |
| 22 | r | cs[2] | code segment for system mode |  |
| 23 | r | cs[3] | code segment for debug mode |  |
| 24 | r | csl[0] | code segment limit for user mode |  |
| 25 | r | csl[1] | code segment limit for user mode |  |
| 26 | r | csl[2] | code segment limit for system mode |  |
| 27 | r | csl[3] | code segment limit for debug mode |  |
| 28 |  | es[0] |  |  |
| 29 |  | es[1] |  |  |
| 2A |  | es[2] |  |  |
| 2B |  | es[3] |  |  |
| 2C |  | esl[0] |  |  |
| 2D |  | esl[1] |  |  |
| 2E |  | esl[2] |  |  |
| 2F |  | esl[3] |  |  |
| 30 |  | fs[0] |  |  |
| 31 |  | fs[1] |  |  |
| 32 |  | fs[2] |  |  |
| 33 |  | fs[3] |  |  |
| 34 |  | fsl[0] |  |  |
| 35 |  | fsl[1] |  |  |
| 36 |  | fsl[2] |  |  |
| 37 |  | fsl[3] |  |  |
| 38 |  | gs[0] |  |  |
| 39 |  | gs[1] |  |  |
| 3A |  | gs[2] |  |  |
| 3B |  | gs[3] |  |  |
| 3C |  | gsl[0] |  |  |
| 3D |  | gsl[1] |  |  |
| 3E |  | gsl[2] |  |  |
| 3F |  | gsl[3] |  |  |
| The following registers represent current segment and limit values | | | |  |
| 40 |  | ds | the current data segment |  |
| 41 |  | dsl | the current data segment limit |  |
| 42 |  | cs | the current code segment |  |
| 43 |  | csl | the current code segment limit |  |
| 44 |  | es |  |  |
| 45 |  | esl |  |  |
| 46 |  | fs |  |  |
| 47 |  | fsl |  |  |
| 48 |  | gs |  |  |
| 49 |  | gsl |  |  |
| 4A |  | ssl | current stack lower bound |  |
| 4B |  | ssu | current stack upper bound |  |
| The following registers used in processing exceptions. | | | |  |
| 50 |  | fault\_pc | program counter value at which fault occurred |  |
| 51 |  | fault\_cs | code segment value at which fault occurred |  |
| 52 |  | fault\_addr | offset value where data fault occurred (load / store) |  |
| 53 |  | fault\_seg | segment value where data fault occurred (load/store) |  |

Example

Return a value from a system service routine to user mode.

mov usr:r1,r1

Send parameters to system routine from user mode

mov sys:r1,r1

mov sys:r2,r2

### RTI – Return from Interrupt

**Description:**

This instruction is used to return from interrupt or exception processing code including external hardware interrupts, internal exceptions, and debug exceptions.

The machine status, program counter, code segment and code segment limit are loaded from the stack. The specified register is transferred to the target register in the mode returned to. This allows the RTI instruction to return a value. Typically this register will be zero however.

**Instruction Format:**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 31 27 | 26 23 | 22 20 | 19 15 | 14 13 | 12 10 | 9 5 | 4 0 |
| 13h5 | ~4 | ~3 | Rt5 | ~2 | ~3 | Ra5 | 02h5 |

**Clock Cycles:** 3 + 5

**Operation:**

**Exceptions:** DBE, DBG, TLB, PRIV

### SEG – Set Segment

**Description:**

This instruction sets the segment register indicator bits to the value specified. (Loads the top three bits of a register with an immediate value).

**Instruction Format:**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 31 27 | 26 20 | 19 15 | 1413 | 12 10 | 9 5 | 4 0 |
| 1Dh5 | ~7 | Rt5 | ~2 | Imm3 | Ra5 | 02h5 |

**Clock Cycles:** 3

**Operation:**

**Rt = {imm3,Ra29..0}**

**Exceptions:** none

# Opcode Map

## Major Opcode

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | x0 | x1 | x2 | x3 | x4 | x5 | x6 | x7 | x8 | x9 | xA | xB | xC | xD | xE | xF |
| 0x | BRK | NOP | {rr} | Bcc | ADDI |  | CMPI |  | ANDI | ORI | EORI |  |  |  |  |  |
| 1x | JMP | JSR |  |  | JMP far |  |  |  |  |  |  |  | LD | LDAR | ST | STCR |

## Function for {rr}

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | x0 | x1 | x2 | x3 | x4 | x5 | x6 | x7 | x8 | x9 | xA | xB | xC | xD | xE | xF |
| 0x | NOT | NEG | COM | MOV | ADD | SUB | CMP |  | AND | OR | EOR | ANDC | NAND | NOR | ENOR | ORC |
| 1x | JMP | JSR | RTS | RTI | SHLI | SHRI | SHL | SHR | ASRI | ASR | MTSPR | MFSPR | DBG | SEG | EXS | EXL |