FISA64

## General Purpose Register Array

There is an array of 32, 64 bit general purpose integer registers.

|  |  |
| --- | --- |
|  | Usage |
| r0 | always zero |
| r1 | return value |
| r2 |  |
| r3 | temporary register |
| r4 | temporary register |
| r5 | temporary register |
| r6 | temporary register |
| r7 | temporary register |
| r8 | temporary register |
| r9 | temporary register |
| r10 | temporary register |
| r11 | register var |
| r12 | register var |
| r13 | register var |
| r14 | register var |
| r15 | register var |
| r16 | register var |
| r17 | register var |
| r18 | register var |
| r19 |  |
| r20 |  |
| r21 |  |
| r22 |  |
| r23 |  |
| r24 | task register (TR) |
| r25 | thread pointer |
| r26 | global pointer |
| r27 | frame pointer (BP) |
| r28 | catch link address (XLR) |
| r29 |  |
| r30 | stack pointer (SP) |
| r31 | return address (LR) |

# Special Purpose Registers

Special purpose registers are accessible only in kernel mode.

## Control Register Zero (SPR 00 or CR0)

This register contains a bit to enable protected mode.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 63 | 62 9 6 1 | | | | | | | | 0 |
| ~ | ~ |  |  |  |  |  | ~ |  | Pe |

PE: Protected Mode enable: 1 = enabled, 0 = disabled.

## Tick Count Register (SPR 04 or TICK)

This register contains a count of the number of clock cycles that have passed since the last time the processor was reset. Tick may be used for high-resolution timing or performance measurement.

## Interrupt Vector Table Base Address (SPR 10 or VBR)

This register contains the physical base address of the interrupt vector table in memory. The Table is 8kB aligned.

|  |  |
| --- | --- |
| 63 13 | 12 0 |
| Address63..13 | 00013 |

Interrupt vector table entries are two 64 bit words in size.

|  |
| --- |
| 63 0 |
| Address 63..0 |
| Reserved64 |

## Code Segment Base (SPR 32)

This register contains the base address of program code in memory. Subroutine calls and jumps are offset from the code base register. Only the offset from the code base is saved in the linkage register during a subroutine call. In user mode there is no way for the program to determine it’s real physical address.

## LOTGRP (SPR 42)

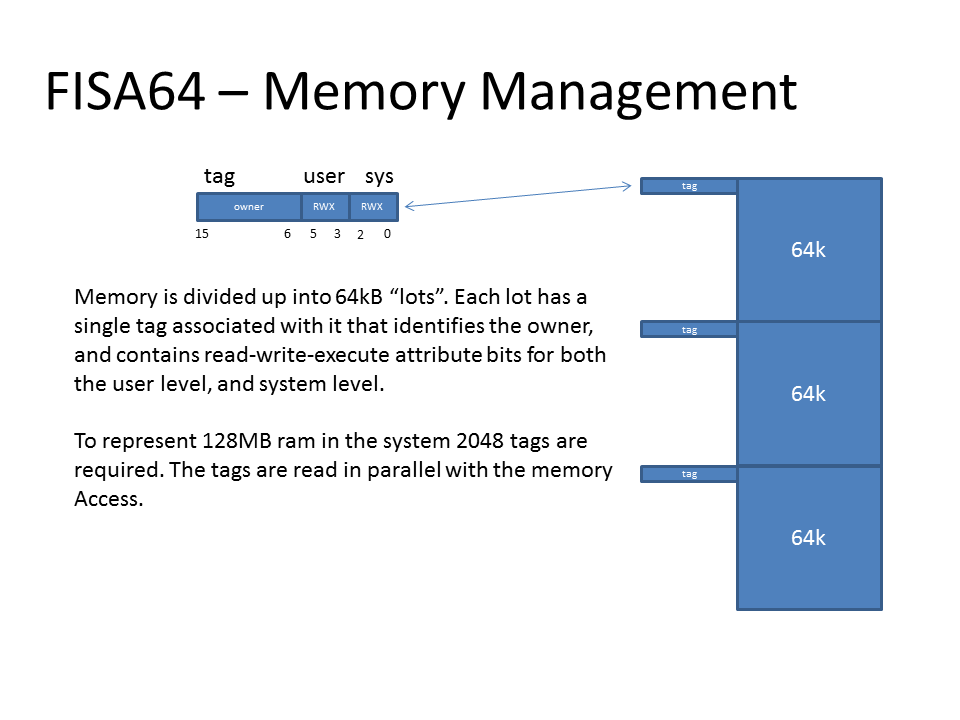
This register contains a list of memory groups that the process belongs to. The owning group associated with a memory tag is compared to this list during a memory access. If the group is in the list then the memory access is allowed, otherwise a memory fault exception occurs. This comparison takes place only in user mode; in kernel mode the kernel owns all of memory so the memory access is always allowed.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 63 60 | 59 50 | 49 40 | 39 30 | 29 20 | 19 10 | 9 0 |
| ~ | Group5 | Group4 | Group3 | Group2 | Group1 | Group0 |

# Memory Protection System

A key feature required to increase system reliability and robustness is memory protection. Memory should be protected against inadvertent access by the process that doesn’t own a particular piece of memory. The system used here provides memory protection, but not address virtualization.

Memory is organized into lots which are 64kB in size. Memory is protected using a system of tags associated with each lot of memory. The tag associated with a memory lot contains the lot owner’s group, and read / write / execute indicators.



The lot owner field in the memory tag represents a group of processes which may access the memory lot. Each process in the system may be associated with up to six memory groups. Which memory groups the process is a part of is stored in the LOTGRP special purpose register.

Code segmentation is used in order to improve the efficiency of jumps and calls (JMP, JSR) and increase the potential address range of the processor. The JMP and JSR instruction specify a 27 bit offset into the code segment. One alternative would be to use program counter relative addressing for jumps and calls. Another alternative would be to use a register based jump where the register is loaded with immediate instructions prior to the jump. This both increases code size and decreases performance.

# Sample Code

## Setting up the code base

The code base register must be setup before a control transfer instruction can be executed.

; The code base register may not need to be saved to the task control block as it’s value

; assigned by the OS shouldn’t change.

MFSPR r1,CDBASE ; get the current code base register

SW r1,varProcCodeBase[tr] ; save it in the task control block

; The OS code base value in use may equal zero, in which case all that is needed is:

; MTSPR CDBASE,r0

LW r1,varOSCodeBase

MTSPR CDBASE,r1 ; store in code base

## Register – Register Format Instructions

FISA64 includes a standard set of arithmetic and logical instructions including add / subtract / multiply/ divide / modulus / logical and / or / and exclusive or. Also present are shift instructions for both signed and unsigned operations.

The CMP instruction performs a signed comparison of two registers, or a register and immediate value and stores a -1, 0, or +1 in the target register if the first operand is less than, equal to or greater than the second operand respectively. The comparison result may be used by a following branch instruction. The CMPU instruction works the same way as CMP except that it performs an unsigned comparison. CMPU performs an unsigned comparison but produces a signed result.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Func7 | ~3 | | Rb5 | | Rt5 | Ra5 | 027 | {RR} |
| 047 |  | | Rb5 | | Rt5 | Ra5 | 027 | ADD |
| 057 |  | | Rb5 | | Rt5 | Ra5 | 027 | SUB |
| 067 |  | | Rb5 | | Rt5 | Ra5 | 027 | CMP |
| 077 |  | | Rb5 | | Rt5 | Ra5 | 027 | MUL |
| 087 |  | | Rb5 | | Rt5 | Ra5 | 027 | DIV |
| 097 |  | | Rb5 | | Rt5 | Ra5 | 027 | MOD |
| 0C7 |  | | Rb5 | | Rt5 | Ra5 | 027 | AND |
| 0D7 |  | | Rb5 | | Rt5 | Ra5 | 027 | OR |
| 0E7 |  | | Rb5 | | Rt5 | Ra5 | 027 | EOR |
| 107 |  | | ~ | | Rt6 | Ra6 | 027 | SXB |
| 117 |  | | ~ | | Rt6 | Ra6 | 027 | SXC |
| 127 |  | | ~ | | Rt6 | Ra6 | 027 | SXH |
| 147 |  | | Rb5 | | Rt5 | Ra5 | 027 | ADDU |
| 157 |  | | Rb5 | | Rt5 | Ra5 | 027 | SUBU |
| 167 |  | | Rb5 | | Rt5 | Ra5 | 027 | CMPU |
| 177 |  | | Rb5 | | Rt5 | Ra5 | 027 | MULU |
| 187 |  | | Rb5 | | Rt5 | Ra5 | 027 | DIVU |
| 197 |  | | Rb5 | | Rt5 | Ra5 | 027 | MODU |
| 1E7 | Spr8 | | | | ~ | Ra5 | 027 | MTSPR |
| 1F7 | Spr8 | | | | Rt5 | ~ | 027 | MFSPR |
| 2x7 |  | | Rb5 | | Rt5 | Ra5 | 027 | Scc |
| 207 |  | | Rb5 | | Rt5 | Ra5 | 027 | SEQ |
| 217 |  | | Rb5 | | Rt5 | Ra5 | 027 | SNE |
| 287 |  | | Rb5 | | Rt5 | Ra5 | 027 | SGT |
| 297 |  | | Rb5 | | Rt5 | Ra5 | 027 | SLE |
| 2A7 |  | | Rb5 | | Rt5 | Ra5 | 027 | SGE |
| 2B7 |  | | Rb5 | | Rt5 | Ra5 | 027 | SLT |
| 2C7 |  | | Rb5 | | Rt5 | Ra5 | 027 | SHI |
| 2D7 |  | | Rb5 | | Rt5 | Ra5 | 027 | SLS |
| 2E7 |  | | Rb5 | | Rt5 | Ra5 | 027 | SHS |
| 2F7 |  | | Rb5 | | Rt5 | Ra5 | 027 | SLO |
| 307 |  | | Rb5 | | Rt5 | Ra5 | 027 | SLL |
| 317 |  | | Rb5 | | Rt5 | Ra5 | 027 | SRL |
| 327 |  | | Rb5 | | Rt5 | Ra5 | 027 | ROL |
| 337 |  | | Rb5 | | Rt5 | Ra5 | 027 | ROR |
| 347 |  | | Rb5 | | Rt5 | Ra5 | 027 | SRA |
| Func7 |  | Imm6 | | | Rt5 | Ra5 | 027 | Shifts # |
| 387 |  | Imm6 | | | Rt5 | Ra5 | 027 | SLL # |
| 397 |  | Imm6 | | | Rt5 | Ra5 | 027 | SRL # |
| 3A7 |  | Imm6 | | | Rt5 | Ra5 | 027 | ROL # |
| 3B7 |  | Imm6 | | | Rt5 | Ra5 | 027 | ROR # |
| 3C7 |  | Imm6 | | | Rt5 | Ra5 | 027 | SRA # |
| 3E7 | ~8 | | | | 05 | ~5 | 027 | RTE |
| 3F7 | ~8 | | | | 05 | ~5 | 027 | RTI |
| 407 | Pred4 | | | Succ4 | 05 | ~5 | 027 | FENCE |

## Register – Immediate Format Instructions

The immediate load instruction loads 16 bit of an immediate value with one operation.

LC0I – loads the low order 16 bits of a register and sign extends the value to 64 bits

LC1I – loads bits 31 to 16 of a register and sign extends the value to 64 bits. The low order 16 bits of the register are not affected.

LC2I – loads bits 47 to 32 of a register and sign extends the value to 64 bits. The low order 32 bits of the register are not affected.

LC3I - loads bits 63 to 48 of a register. The low order 32 bits of the register are not affected.

There are signed and unsigned versions of instructions. The mnemonics of the unsigned instructions are post-fixed with a ‘U’.

ADD / SUB may generate an overflow exception when overflow occurs. ADDU / SUBU do not generate any exceptions.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Immed15 | Rt5 | Ra5 | | 047 | ADD # |
| Immed15 | Rt5 | Ra5 | | 057 | SUB # |
| Immed15 | Rt5 | Ra5 | | 067 | CMP # |
| Immed15 | Rt5 | Ra5 | | 077 | MUL # |
| Immed15 | Rt5 | Ra5 | | 087 | DIV # |
| Immed15 | Rt5 | Ra5 | | 097 | MOD # |
| Immed15 | Rt5 | I1 | O4 | 0A7 | LCn # |
| Immed15 | Rt5 | Ra5 | | 0C7 | AND # |
| Immed15 | Rt5 | Ra5 | | 0D7 | OR # |
| Immed15 | Rt5 | Ra5 | | 0E7 | EOR # |
| Immed15 | Rt5 | Ra5 | | 147 | ADDU # |
| Immed15 | Rt5 | Ra5 | | 157 | SUBU # |
| Immed15 | Rt5 | Ra5 | | 167 | CMPU # |
| Immed15 | Rt5 | Ra5 | | 177 | MULU # |
| Immed15 | Rt5 | Ra5 | | 187 | DIVU # |
| Immed15 | Rt5 | Ra5 | | 197 | MODU # |
| Immed15 | Rt5 | Ra5 | | 2x7 | Scc # |
| Immed15 | Rt5 | Ra5 | | 207 | SEQ # |
| Immed15 | Rt5 | Ra5 | | 217 | SNE # |
| Immed15 | Rt5 | Ra5 | | 287 | SGT # |
| Immed15 | Rt5 | Ra5 | | 297 | SLE # |
| Immed15 | Rt5 | Ra5 | | 2A7 | SGE # |
| Immed15 | Rt5 | Ra5 | | 2B7 | SLT # |
| Immed15 | Rt5 | Ra5 | | 2C7 | SHI # |
| Immed15 | Rt5 | Ra5 | | 2D7 | SLS # |
| Immed15 | Rt5 | Ra5 | | 2E7 | SHS # |
| Immed15 | Rt5 | Ra5 | | 2F7 | SLO # |

## Flow Control Instructions

There are six relational branches which branch based on the result of a signed comparison of a register to zero. In order to branch based on an unsigned comparison, the CMPU instruction must be used prior to the branch. Since branches inherently compare a register to zero it is often possible to omit a preceding compare (CMP) operation. Branches branch relative to the program counter using a 17 bit signed displacement. This allows branching within +/- 64kB of the current program address.

The subroutine call instruction (BSR) stores the return address in the default link register – R31. The target address is specified as a 27 bit displacement from the current program counter.

In order to jump to a routine whose target address is computed in a register at run time, the JAL instruction is provided.

The BRA instruction works the same way as the BSR instruction, but doesn’t store the return address.

The RTS instruction is used to return from a subroutine and de-allocate a stack frame at the same time.

The BRK instruction is used to transfer control to a kernel mode BRK handler. This is the means to communicate with the operating system. Hardware interrupts force an appropriate BRK instruction into the instruction stream.

The NOP instruction doesn’t perform any operation.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Flow Control | | | | | | | |
| Disp16..2 | | | ~2 | 03 | Ra5 | 3D7 | BEQ |
| Disp16..2 | | | ~2 | 13 | Ra5 | 3D7 | BNE |
| Disp16..2 | | | ~2 | 23 | Ra5 | 3D7 | BGT |
| Disp16..2 | | | ~2 | 33 | Ra5 | 3D7 | BGE / BPL |
| Disp16..2 | | | ~2 | 43 | Ra5 | 3D7 | BLT / BMI |
| Disp16..2 | | | ~2 | 53 | Ra5 | 3D7 | BLE |
| H | ~5 | Vector9 | ~5 | | ~5 | 387 | BRK |
| Offset26..2 | | | | | | 397 | BSR |
| Offset26..2 | | | | | | 3A7 | BRA |
| Immed15 | | | Rt5 | | 1F5 | 3B7 | RTS |
| Immed15 | | | Rt5 | | Ra5 | 3C7 | JAL |
| ~25 | | | | | | 3F7 | NOP |

## Memory Operate Instructions

FISA64 is a load / store architecture.

There are two different instruction formats for memory operating instructions. These are register indirect with displacement format and scaled indexed addressing format.

Operand sizes of byte (8 bit), character (16 bit), half-word (32 bit) and word (64 bits) are supported. Sign and zero extension on load is available.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Memory | | | | | | 64 bit | | 32 bit |
| Disp15 | | | Rt5 | Ra5 | 407 | | LB | LB |
| Disp15 | | | Rt5 | Ra5 | 417 | | LBU | LBU |
| Disp15 | | | Rt5 | Ra5 | 427 | | LC | LH |
| Disp15 | | | Rt5 | Ra5 | 437 | | LCU | LHU |
| Disp15 | | | Rt5 | Ra5 | 447 | | LH | LW |
| Disp15 | | | Rt5 | Ra5 | 457 | | LHU | - |
| Disp15 | | | Rt5 | Ra5 | 467 | | LW | - |
| Offs8 | Sc2 | Rb5 | Rt5 | Ra5 | 487 | | LBX | LBX |
| Offs8 | Sc2 | Rb5 | Rt5 | Ra5 | 497 | | LBUX | LBUX |
| Offs8 | Sc2 | Rb5 | Rt5 | Ra5 | 4A7 | | LCX | LHX |
| Offs8 | Sc2 | Rb5 | Rt5 | Ra5 | 4B7 | | LCUX | LHUX |
| Offs8 | Sc2 | Rb5 | Rt5 | Ra5 | 4C7 | | LHX | LWX |
| Offs8 | Sc2 | Rb5 | Rt5 | Ra5 | 4D7 | | LHUX | - |
| Offs8 | Sc2 | Rb5 | Rt5 | Ra5 | 4E7 | | LWX | - |
| Offs8 | Sc2 | Rb5 | Rt5 | Ra5 | 4F7 | | LEAX | LEAX |
| Disp15 | | | Rs5 | Ra5 | 607 | | SB | SB |
| Disp15 | | | Rs5 | Ra5 | 617 | | SC | SH |
| Disp15 | | | Rs5 | Ra5 | 627 | | SH | SW |
| Disp15 | | | Rs5 | Ra5 | 637 | | SW | - |
| Disp15 | | | Imm5 | Ra5 | 647 | | INC | INC |
| Offs8 | Sc2 | Rb5 | Rs5 | Ra5 | 687 | | SBX | SBX |
| Offs8 | Sc2 | Rb5 | Rs5 | Ra5 | 697 | | SCX | SHX |
| Offs8 | Sc2 | Rb5 | Rs5 | Ra5 | 6A7 | | SHX | SWX |
| Offs8 | Sc2 | Rb5 | Rs5 | Ra5 | 6B7 | | SWX | - |

Loads and stores do not have to be aligned, however unaligned access will require additional clock cycles to complete.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | x0 | x1 | x2 | x3 | x4 | x5 | x6 | x7 | x8 | x9 | xA | xB | xC | xD | xE | xF |
| 0x |  |  | {rr} |  | ADD# | SUB# | CMP# | MUL# | DIV# | MOD# | LD# |  | AND# | OR# | EOR# |  |
| 1x |  |  |  |  | ADDU# | SUBU# | CMPU # | MULU# | DIVU# | MODU# |  |  |  |  |  |  |
| 2x | SEQ# | SNE# |  |  |  |  |  |  | SGT# | SLE# | SGE# | SLT# | SHI# | SLS# | SHS# | SLO# |
| 3x |  |  |  |  |  |  |  |  | BRK | JSR | JMP | RTS | JAL | Bcc |  | NOP |
| 4x | LB | LBU | LC | LCU | LH | LHU | LW | LEA | LBX | LBUX | LCX | LCUX | LHX | LHUX | LWX | LEAX |
| 5x | LFS | LFD | LFQ |  |  |  |  |  | LFSX | LFDX | LFQX |  |  |  |  |  |
| 6x | SB | SC | SH | SW | INC |  |  |  | SBX | SCX | SHX | SWX |  |  |  |  |
| 7x | SFS | SFD | SFQ |  |  |  |  |  | SFSX | SFDX | SFQX |  | IMM0 | | IMM1 | |

02 Group Func

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | x0 | x1 | x2 | x3 | x4 | x5 | x6 | x7 | x8 | x9 | xA | xB | xC | xD | xE | xF |
| 0x | NAND | NOR | ENOR |  | ADD | SUB | CMP | MUL | DIV | MOD |  |  | AND | OR | EOR |  |
| 1x | SXB | SXC | SXH |  | ADDU | SUBU | CMPU | MULU | DIVU | MODU |  |  |  |  | MTSPR | MFSPR |
| 2x | SEQ | SNE |  |  |  |  |  |  | SGT | SLE | SGE | SLT | SHI | SLS | SHS | SLO |
| 3x | SLL | SRL | ROL | ROR | SRA |  |  |  | SLLI | SRLI | ROLI | RORI | SRAI |  | RTE | RTI |
| 4x | FENCE |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 5x |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 6x | FADD | FSUB | FMUL | FDIV |  |  |  |  |  |  |  |  |  |  |  |  |
| 7x |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |