### RF65000

Register File

The register file contains 32 general purpose registers.

The register file is a *unified* register file. Registers may contain integer or floating-point data.

|  |  |  |  |
| --- | --- | --- | --- |
| Regno | ABI | ABI Usage |  |
| 0 | 0 | Always zero, for indexed addressing modes, refers to the program counter |  |
| 1 | A0 | First argument / return value register |  |
| 2 | A1 | Second argument / return value register |  |
| 3 | T0 | Temporary register, caller save |  |
| 4 | T1 | Temporary register |  |
| 5 | T2 | Temporary register |  |
| 6 | T3 | Temporary register |  |
| 7 | T4 | Temporary register |  |
| 8 | T5 | Temporary register |  |
| 9 | T6 | Temporary register |  |
| 10 | T7 | Temporary register |  |
| 11 | S0 | Saved register, register variables |  |
| 12 | S1 | Saved register |  |
| 13 | S2 | Saved register |  |
| 14 | S3 | Saved register |  |
| 15 | S4 | Saved register |  |
| 16 | S5 | Saved register |  |
| 17 | S6 | Saved register |  |
| 18 | S7 | Saved register |  |
| 19 | S8 | Saved register |  |
| 20 | A2 | Third argument register |  |
| 21 | A3 | Argument register |  |
| 22 | A4 | Argument register |  |
| 23 | A5 | Argument register |  |
| 24 | A6 | Argument register |  |
| 25 | A7 | Argument register |  |
| 26 |  |  |  |
| 27 | GP2 | Global pointer – rodata section |  |
| 28 | GP1 | Global pointer – bss section |  |
| 29 | GP0 | Global pointer – data section |  |
| 30 | FP | Frame Pointer |  |
| 31 | SP | Stack pointer |  |
| 31 | USP | User stack pointer |  |
| 31 | SSP | System stack pointer |  |
|  | LC | Loop counter |  |
|  | PC | Program counter |  |
|  | CCRG | Condition Codes Register Group |  |
|  | AC | Application Control Register |  |
|  | SR | Status Register |  |

### CC - Condition Codes Register Group

There is a set of eight condition codes registers which may be referenced individually or together in a group called the CCRG. A condition register holds four flags which reflect the status of operations. Most instructions can select a condition register to update.

**Condition Register Group Register**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 31 28 | 27 24 | 23 20 | 19 16 | 15 12 | 11 8 | 7 4 | 3 0 |
| Cr7 | Cr6 | Cr5 | Cr4 | Cr3 | Cr2 | Cr1 | Cr0 |

**Condition Register Format:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 3 | 2 | 1 | 0 |
| Integer | nf | zf | vf | cf |
| Float | lt | eq | ord | ~ |

### SR - Status Register

The processor status register holds bits controlling the overall operation of the processor. The status register is not accessible in user mode.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 31 24 | 23 21 | 20 16 | 15 |  | 13 | 12 | 11 | 10 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| CPL | ~ | ~ | T |  | S |  |  | IPL |  |  |  |  |  | N | RT | |

CPL is the current privilege level the processor is operating at.

T indicates that trace mode is active.

S indicates the processor is in supervisor mode.

N indicates that near or short (32-bit) addressing is in use. When short addressing is in use only the low order 32-bit are significant and stored or loaded to or from the stack.

IPL is the interrupt mask level

RT specifies the return type for an [RTI](#_RTI_–_Return) instruction.

### AC – Application Control Register

This register holds the address of the applications vector table and three bits indicating the use of decimal mode by the application. The vector table must be 256-byte aligned.

|  |  |  |  |
| --- | --- | --- | --- |
| 63 8 | 7 3 | 2 | 1 0 |
| App Vector Table Address63..8 | ~ | D | D |

#### Decimal Mode

Setting the ‘d’ flag bit 2 in the AC register sets the processor in decimal operating mode. Arithmetic operations will use BCD numbers for both source and destination operands.

Decimal mode, ‘d’ flag bit 0 and 1, may also be applied to floating-point which will use decimal floating-point operations instead of binary.

### VB – Vector Base Register

The vector base register provides the location of the vector table. The vector table must be 2kB aligned. On reset the VBR is loaded with zero.

|  |  |  |  |
| --- | --- | --- | --- |
| 63 11 | 10 3 | 2 | 1 0 |
| Vector Table Address63..11 | ~ | ~ | ~ |

## Exceptions

Exception Table

|  |  |
| --- | --- |
| Vector | Usage |
| 0 | Reset value for system stack pointer |
| 1 | Reset value for program counter |
| 2 | Bus Error |
| 3 | Address Error |
| 4 | Unimplemented Instruction |
| 5 |  |
| 6 |  |
| 7 |  |
| 8 | Privilege Violation |
| 9 | Instruction trace |
| 10 |  |
| 11 to 23 | reserved |
| 24 | Spurious interrupt |
| 25 | Auto vector #1 |
| 26 | Auto vector #2 |
| 27 | Auto vector #3 |
| 28 | Auto vector #4 |
| 29 | Auto vector #5 |
| 30 | Auto vector #6 |
| 31 | Auto vector #7 |
| 32 | Breakpoint (BRK) |
| 33 to 63 | Trap #1 to 31 |
| 64 to 255 | User usage |
|  |  |

Application Exception Table

|  |  |
| --- | --- |
| Vector | Usage |
| 0 | Divide by zero |
| 1 to 30 | Trap #1 to #30 |
| 31 | Overflow |

# Instruction Descriptions

### Major Opcode

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| 0x | 0  TRAP | 1  {R1} | 2  {CSR} | 3  CMP / FCMP | 4  ADD | 5  SUB | 6  MUL | 7  DIV |
|  | 8  AND | 9  OR | 10  XOR | 11 | 12 | 13  {BIT} | 14  SHIFT | 15  {Misc} |
| 1x | 16  LOAD | 17  LOADM | 18  STORE | 19  STOREM | 20  FMA | 21  FMS | 22  FNMA | 23  FNMS |
|  | 24  JSR / JMP | 25  LOADQ | 26 | 27  PEA | 28  Bcc | 29  DBcc | 30 | 31 |

Operand Swapping

Most instructions allow first and second source operands to be swapped. This is indicated by the swap ‘S’ bit in the instruction.

**Load and Store Address Modes**

|  |  |  |  |
| --- | --- | --- | --- |
| **Address Mode** | **Format** | **M2** | **Reg5** |
| Indexed1 with displacement | Disp(Rn, Rn) | 0 | Rn |
| Post Update indexed with displacement | Disp(Rn,Rn+) | 1 | Rn |
| Pre Update indexed indirect with displacement | Disp(+Rn,Rn) | 2 | Rn |
| Immediate Eleven | Imm11 | 3 | nnnnn |
| Immediate Thirty-Two | Imm32 | 3 | nnnnn |
| Immediate Sixty-Four | Imm64 | 3 | nnnnn |

1For indexed modes, R0 refers to the program counter

**Operation Size**

|  |  |  |
| --- | --- | --- |
| **Operation Size** | **Suffix** | **Sz2** |
| Byte | .b | 00 |
| Wyde | .w | 01 |
| Tetra | .t | 10 |
| Octa | .o | 11 |

Operand Swap

|  |  |
| --- | --- |
| **Operand Order** | **S** |
| Normal | 0 |
| 1st and 2nd Swapped | 1 |

### Operand Sizes

Many instructions support four different operand sizes: byte, wyde, tetra and octa. The operand size is selected by suffixing the mnemonic with ‘b’ for byte, ‘w’ for wyde, ‘t’ for tetra and ‘o’ for octa.

|  |  |
| --- | --- |
| Sz2 | Operand Size |
| 00 | Byte |
| 01 | Wyde |
| 10 | Tetra |
| 11 | Octa |

## Arithmetic Operations

### ABS – Absolute Value

**Description:**

This instruction computes the absolute value of the contents of the source operand and places the result in Rt.

**Integer Instruction Format: R1**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 31 27 | 26 | 25 21 | 20 18 | 17 13 | 12 8 | 7 6 | 5 0 |
| Rc5 | 1 | Rb5 | 03 | 65 | Rt5 | Sz2 | 16 |

**Supported Address Modes:**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Rn | D(Rn) | D(Rn)+ | -D(Rn) | (Rn, Rn) | D32(Rn,Rn) | D64(Rn,Rn) | Imm |
| \* | \* | \* | \* | \* | \* | \* |  |

**Operation:**

If <ea> < 0

Rt = -<ea>

else

Rt = <ea>

**Execution Units:** Integer ALU #0

**Clock Cycles: 1**

**Exceptions:** none

**Notes:**

### ADC – Add with Carry

**Description:**

Add two source operands and carry and place the sum in the target register. All registers are integer registers. Arithmetic is signed twos-complement values unless the decimal mode flag is set in which case values are treated as BCD numbers.

**Supported Operand Sizes:** .b, .w, .t, .o

**Flag Updates:**

Condition Register Ct3 is always updated.

If the result is non-zero the zero flag is cleared, otherwise the zero flag is unchanged.

If the result is negative the negative flag is set, otherwise it is cleared.

If there is a carry the carry flag is set in the condition register, otherwise it is cleared.

If there is an overflow, the overflow flag is set in the condition register, otherwise it is cleared.

**Operation:**

Rt = Ra + Rb + c or Rt = Ra + Imm + c

**Clock Cycles:**

**Execution Units:** All Integer ALU’s

**Exceptions:** none

**Notes:**

**Instruction Formats:**

**ADC Rt, Ra, Rb – Register direct**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 31 29 | 28 24 | 23 | 22 18 | 17 | 16 12 | 11 7 | 6 5 | 4 0 |
| Ct3 | ~5 | 1 | Rb5 | 01 | Ra5 | Rt5 | Sz2 | 45 |

**Clock Cycles: 4**

**ADC Rt,Ra,Imm10**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 31 29 | 28 24 | 23 | 22 18 | 17 | 16 12 | 11 7 | 6 5 | 4 0 |
| Ct3 | Imm9..5 | 1 | Imm4..0 | 11 | Ra5 | Rt5 | Sz2 | 45 |

**Clock Cycles: 4**

**ADC Rt,Ra,Imm32**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 31 29 | 28 24 | 23 | 22 18 | 17 | 16 12 | 11 7 | 6 5 | 4 0 |
| Ct3 | 165 | 1 | ~5 | 11 | Ra5 | Rt5 | Sz2 | 45 |
| Immediate32 | | | | | | | | |

**Clock Cycles: 7**

**ADC Rt,Ra,Imm64**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 31 29 | 28 24 | 23 | 22 18 | 17 | 16 12 | 11 7 | 6 5 | 4 0 |
| Ct3 | 175 | 1 | ~5 | 11 | Ra5 | Rt5 | Sz2 | 45 |
| Immediate31..0 | | | | | | | | |
| Immediate63..32 | | | | | | | | |

**Clock Cycles: 10**

### ADD - Addition

**Description:**

Add two source operands and place the sum in the target register. All registers are integer registers. Arithmetic is signed twos-complement values unless the decimal mode flag is set in which case values are treated as BCD numbers.

**Supported Operand Sizes:** .b, .w, .t, .o

**Flag Updates:**

Condition Register Ct3 is always updated.

If the result is zero the zero flag is set otherwise it is cleared.

If the result is negative the negative flag is set, otherwise it is cleared.

If there is a carry the carry flag is set in the condition register, otherwise it is cleared.

If there is an overflow, the overflow flag is set in the condition register, otherwise it is cleared.

**Operation:**

Rt = Ra + Rb or Rt = Ra + Imm

**Clock Cycles:**

**Execution Units:** All Integer ALU’s

**Exceptions:** none

**Notes:**

**Instruction Formats:**

**ADD Rt, Ra, Rb – Register direct**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 31 29 | 28 24 | 23 | 22 18 | 17 | 16 12 | 11 7 | 6 5 | 4 0 |
| Ct3 | ~5 | 0 | Rb5 | 01 | Ra5 | Rt5 | Sz2 | 45 |

**Clock Cycles: 4**

**ADD Rt,Ra,Imm10**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 31 29 | 28 24 | 23 | 22 18 | 17 | 16 12 | 11 7 | 6 5 | 4 0 |
| Ct3 | Imm9..5 | 0 | Imm4..0 | 11 | Ra5 | Rt5 | Sz2 | 45 |

**Clock Cycles: 4**

**ADD Rt,Ra,Imm32**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 31 29 | 28 24 | 23 | 22 18 | 17 | 16 12 | 11 7 | 6 5 | 4 0 |
| Ct3 | 165 | 0 | ~5 | 11 | Ra5 | Rt5 | Sz2 | 45 |
| Immediate32 | | | | | | | | |

**Clock Cycles: 7**

**ADD Rt,Ra,Imm64**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 31 29 | 28 24 | 23 | 22 18 | 17 | 16 12 | 11 7 | 6 5 | 4 0 |
| Ct3 | 175 | 0 | ~5 | 11 | Ra5 | Rt5 | Sz2 | 45 |
| Immediate31..0 | | | | | | | | |
| Immediate63..32 | | | | | | | | |

**Clock Cycles: 10**

### AND – Bitwise And

**Description:**

Bitwise ‘and’ two source operands and place the sum in the target register. All registers are integer registers. Arithmetic is signed twos-complement values.

**Supported Operand Sizes:** .b, .w, .t, .o

**Flag Updates:**

Condition Register Ct3 is always updated.

If the result is zero the equals flag is set otherwise it is cleared.

If the result is negative the signed less than flag is set, otherwise it is cleared.

**Operation:**

Rt = Ra & Rb or Rt = Ra & Imm

**Instruction Formats:**

**AND Rt, Ra, Rb – Register direct**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 31 29 | 28 24 | 23 | 22 18 | 17 | 16 12 | 11 7 | 6 5 | 4 0 |
| Ct3 | ~5 | 0 | Rb5 | 01 | Ra5 | Rt5 | Sz2 | 85 |

**Clock Cycles: 4**

**AND Rt,Ra,Imm10**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 31 29 | 28 24 | 23 | 22 18 | 17 | 16 12 | 11 7 | 6 5 | 4 0 |
| Ct3 | Imm9..5 | 0 | Imm4..0 | 11 | Ra5 | Rt5 | Sz2 | 85 |

**Clock Cycles: 4**

**AND Rt,Ra,Imm32**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 31 29 | 28 24 | 23 | 22 18 | 17 | 16 12 | 11 7 | 6 5 | 4 0 |
| Ct3 | 165 | 0 | ~5 | 11 | Ra5 | Rt5 | Sz2 | 85 |
| Immediate32 | | | | | | | | |

**Clock Cycles: 7**

**AND Rt,Ra,Imm64**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 31 29 | 28 24 | 23 | 22 18 | 17 | 16 12 | 11 7 | 6 5 | 4 0 |
| Ct3 | 175 | 0 | ~5 | 11 | Ra5 | Rt5 | Sz2 | 85 |
| Immediate31..0 | | | | | | | | |
| Immediate63..32 | | | | | | | | |

**Clock Cycles: 10**

**Clock Cycles:**

**Execution Units:** All Integer ALU’s

**Exceptions:** none

**Notes:**

### CMP - Comparison

**Description:**

Compare two source operands and place the result in the target condition register. The result is a vector identifying the relationship between the two source operands as signed and unsigned integers.

**Supported Operand Sizes:** .b, .w, .t, .o

**Condition Register Format:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 3 | 2 | 1 | 0 |
| Integer | nf | zf | vf | cf |
| Float | lt | eq | ord | ~ |

**Related Branch Condition:**

|  |  |  |
| --- | --- | --- |
| **Cond5** | **Mnemonic** | **Test** |
| 0 | RA | 1 |
| 1 | SR | 0 |
| 2 | HI | !cf & !zf |
| 3 | LS | cf | zf |
| 4 | CC / HS | !cf |
| 5 | CS / LO | cf |
| 6 | NE | !zf |
| 7 | EQ | zf |
| 8 | VC | !vf |
| 9 | VS | vf |
| A | PL | !nf |
| B | MI | nf |
| C | GE | (nf & vf) | (!nf & !vf) |
| D | LT | (nf & !vf) | (!nf & vf) |
| E | GT | (nf & vf & !zf) | (!nf & !vf & zf) |
| F | LE | zf | (nf & !vf) | (!nf & vf) |
| 10 | FOR | ord |
| 11 | FUN | !ord |
| 12 | FGT | !(lt | eq) |
| 13 | FLE | lt | eq |
| 14 | FGE | !lt |
| 15 | FLT | lt |
| 16 | FNE | !eq |
| 17 | FEQ | eq & ord |
| 18 | - |  |
| 19 | - |  |
| 1A | - |  |
| 1B | - |  |
| 1C | - |  |
| 1D | - |  |
| 1E | - |  |
| 1F | - |  |

**Operation:**

Ct = Ra ? Rb or Ct = Ra ? Imm or Ct = Imm ? Ra

**Clock Cycles:**

**Execution Units:** All Integer ALU’s

**Exceptions:** none

**Notes:**

**Instruction Formats:**

**CMP Ct, Ra, Rb – Register direct**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 31 29 | 28 24 | 23 | 22 18 | 17 | 16 12 | 11 8 | 7 | 6 5 | 4 0 |
| Ct3 | ~5 | 0 | Rb5 | 01 | Ra5 | ~4 | 0 | Sz2 | 35 |

**CMP Ct,Ra,Imm10**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 31 29 | 28 24 | 23 | 22 18 | 17 | 16 12 | 11 8 | 7 | 6 5 | 4 0 |
| Ct3 | Imm9..5 | S | Imm4..0 | 11 | Ra5 | ~4 | 0 | Sz2 | 35 |

**CMP Ct,Ra,Imm32**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 31 29 | 28 24 | 23 | 22 18 | 17 | 16 12 | 11 8 | 7 | 6 5 | 4 0 |
| Ct3 | 165 | S | ~5 | 11 | Ra5 | ~4 | 0 | Sz2 | 35 |
| Immediate32 | | | | | | | | | |

**CMP Ct,Ra,Imm64**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 31 29 | 28 24 | 23 | 22 18 | 17 | 16 12 | 11 8 | 7 | 6 5 | 4 0 |
| Ct3 | 175 | S | ~5 | 11 | Ra5 | ~4 | 0 | Sz2 | 35 |
| Immediate31..0 | | | | | | | | | |
| Immediate63..32 | | | | | | | | | |

### CSR – Control and Special Registers Operations

**Description:**

Perform an operation on a CSR. A CSR field value of zero indicates to use the next instruction word as the CSR register number.

|  |  |  |
| --- | --- | --- |
| **Operation** | **Op3** |  |
| MOVE from CSR | 0 |  |
| MOVE to CSR | 1 |  |
| Or to CSR (set bits) | 2 |  |
| And complement to CSR (clear bits) | 3 |  |
| Exclusive Or to CSR (flip bits) | 4 |  |

**Supported Operand Sizes:** .o

|  |  |  |
| --- | --- | --- |
| **Regno** |  |  |
| 0 | {ext} | Extended CSR number |
| 1 | ccr | Condition codes |
| 2 | sr | Status register (privileged) |
| 120 | Tick | Tick count (read only) |
| 121 | Coreno | Core number ( read only) (privileged) |
| 127 |  |  |

**Instruction Formats:**

**MOVE CSR, <ea>**

**OR CSR,<ea>**

**ANDC CSR,<ea>**

**EOR CSR, <ea>**

**MOVE <ea>,CSR**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 31 27 | 26 | 25 21 | 20 18 | 17 11 | 10 8 | 7 6 | 5 0 |
| Rc5 | ~ | Rb5 | 03 | CSRt7 | Op3 | 32 | 26 |

### DIVS – Signed Division

**Description:**

Divide source dividend operand by divisor operand and place the quotient in the target register. All registers are integer registers. Arithmetic is signed twos-complement values.

For register direct mode, the remainder is placed in register Rr. Otherwise, the remainder is not available.

**Supported Operand Sizes:** .b, .w, .t, .o

**Flag Updates:**

Condition Register Ct3 is always updated.

If the result is zero the zero flag is set otherwise it is cleared.

If the result is negative the negative flag is set, otherwise it is cleared.

The carry flag is cleared.

If there is an overflow, the overflow flag is set in the condition register, otherwise it is cleared.

**Operation:**

Rt = Ra / Rb or Rt = Ra / Imm or Rt = Imm / Ra

**Instruction Formats:**

**DIVS Rt, Ra, Rb, Rr – Register direct**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 31 29 | 28 24 | 23 | 22 18 | 17 | 16 12 | 11 7 | 6 5 | 4 0 |
| Ct3 | Rr5 | 0 | Rb5 | 01 | Ra5 | Rt5 | Sz2 | 75 |

**DIVS Rt,Ra,Imm10**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 31 29 | 28 24 | 23 | 22 18 | 17 | 16 12 | 11 7 | 6 5 | 4 0 |
| Ct3 | Imm9..5 | S | Imm4..0 | 11 | Ra5 | Rt5 | Sz2 | 75 |

**DIVS Rt,Ra,Imm32**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 31 29 | 28 24 | 23 | 22 18 | 17 | 16 12 | 11 7 | 6 5 | 4 0 |
| Ct3 | 165 | S | 05 | 11 | Ra5 | Rt5 | Sz2 | 75 |
| Immediate32 | | | | | | | | |

**DIVS Rt,Ra,Imm64**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 31 29 | 28 24 | 23 | 22 18 | 17 | 16 12 | 11 7 | 6 5 | 4 0 |
| Ct3 | 175 | S | 05 | 11 | Ra5 | Rt5 | Sz2 | 75 |
| Immediate31..0 | | | | | | | | |
| Immediate63..32 | | | | | | | | |

**Clock Cycles:** 70

**Execution Units:** All Integer ALU’s

**Exceptions:** none

**Notes:**

### DIVU – Unsigned Division

**Description:**

Divide source dividend operand by divisor operand and place the sum in the target register. All registers are integer registers. Arithmetic is unsigned twos-complement values.

For register direct mode, the remainder is placed in register Rr. Otherwise, the remainder is not available.

Ten-bit immediate mode is not available for this instruction.

**Supported Operand Sizes:** .b, .w, .t, .o

**Flag Updates:**

Condition Register Ct3 is always updated.

If the result is zero the zero flag is set otherwise it is cleared.

If the result is negative the negative flag is set, otherwise it is cleared.

The carry flag is cleared.

If there is an overflow, the overflow flag is set in the condition register, otherwise it is cleared.

**Operation:**

Rt = Ra / Rb or Rt = Ra / Imm or Rt = Imm / Ra

**Instruction Formats:**

**DIVU Rt, Ra, Rb – Register direct**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 31 29 | 28 24 | 23 | 22 18 | 17 | 16 12 | 11 7 | 6 5 | 4 0 |
| Ct3 | Rr5 | 1 | Rb5 | 01 | Ra5 | Rt5 | Sz2 | 75 |

**DIVU Rt,Ra,Imm32**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 31 29 | 28 24 | 23 | 22 18 | 17 | 16 12 | 11 7 | 6 5 | 4 0 |
| Ct3 | 165 | S | 15 | 11 | Ra5 | Rt5 | Sz2 | 75 |
| Immediate32 | | | | | | | | |

**DIVU Rt,Ra,Imm64**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 31 29 | 28 24 | 23 | 22 18 | 17 | 16 12 | 11 7 | 6 5 | 4 0 |
| Ct3 | 175 | S | 15 | 11 | Ra5 | Rt5 | Sz2 | 75 |
| Immediate31..0 | | | | | | | | |
| Immediate63..32 | | | | | | | | |

**Clock Cycles:** 70

**Execution Units:** All Integer ALU’s

**Exceptions:** none

**Notes:**

### EOR – Bitwise Exclusive Or

**Description:**

Bitwise exclusive ‘or’ two source operands and place the sum in the target register. All registers are integer registers. Arithmetic is signed twos-complement values.

**Supported Operand Sizes:** .b, .w, .t, .o

**Flag Updates:**

Condition Register Ct3 is always updated.

If the result is zero the equals flag is set otherwise it is cleared.

If the result is negative the signed less than flag is set, otherwise it is cleared.

**Operation:**

Rt = Ra ^ Rb or Rt = Ra ^ Imm

**Instruction Formats:**

**EOR Rt, Ra, Rb – Register direct**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 31 29 | 28 24 | 23 | 22 18 | 17 | 16 12 | 11 7 | 6 5 | 4 0 |
| Ct3 | ~4 | 0 | Rb5 | 01 | Ra5 | Rt5 | Sz2 | 105 |

**EOR Rt,Ra,Imm10**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 31 29 | 28 24 | 23 | 22 18 | 17 | 16 12 | 11 7 | 6 5 | 4 0 |
| Ct3 | Imm9..5 | 0 | Imm4..0 | 11 | Ra5 | Rt5 | Sz2 | 105 |

**EOR Rt,Ra,Imm32**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 31 29 | 28 24 | 23 | 22 18 | 17 | 16 12 | 11 7 | 6 5 | 4 0 |
| Ct3 | 165 | 0 | ~5 | 11 | Ra5 | Rt5 | Sz2 | 105 |
| Immediate32 | | | | | | | | |

**EOR Rt,Ra,Imm64**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 31 29 | 28 24 | 23 | 22 18 | 17 | 16 12 | 11 7 | 6 5 | 4 0 |
| Ct3 | 175 | 0 | ~5 | 11 | Ra5 | Rt5 | Sz2 | 105 |
| Immediate31..0 | | | | | | | | |
| Immediate63..32 | | | | | | | | |

**Clock Cycles:** 2

**Execution Units:** All Integer ALU’s

**Exceptions:** none

**Notes:**

### LOADQ – Load Quick Immediate

**Description:**

The loadq instruction loads an immediate constant into a register. To load a constant larger than seventeen bits, use the [OR](#_OR_–_Bitwise) or [ADD](#_ADD_-_Addition) instruction.

**Supported Operand Sizes:** .b, .w, .t, .o

**Flag Updates:**

Condition Register Ct3 is always updated.

If the result is zero the zero flag is set otherwise it is cleared.

If the result is negative the negative flag is set, otherwise it is cleared.

**Instruction Format:**

**LOADQ Rt,Imm17**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 31 29 | 28 12 | 11 7 | 6 5 | 4 0 |
| Ct3 | Immediate17 | Rt5 | Sz2 | 255 |

### MULS – Multiply Signed

**Description:**

Multiply two source operands and place the sum in the target register. All registers are integer registers. Arithmetic is signed twos-complement values. The ‘S’ flag indicates to perform an unsigned multiply.

The high order product bits may be placed in register Rp for the register direct form of the instruction.

**Supported Operand Sizes:** .b, .w, .t, .o

**Flag Updates:**

Condition Register Ct3 is always updated.

If the result is zero the zero flag is set otherwise it is cleared.

If the result is negative the negative flag is set, otherwise it is cleared.

If there is a carry the carry flag is set in the condition register, otherwise it is cleared.

If there is an overflow, the overflow flag is set in the condition register, otherwise it is cleared.

Overflow for an unsigned multiply is detected when the product bits 64 to 127 are non-zero. For signed multiply overflow is detected as product bits 64 to 127 not equalling bit 63, the sign bit.

**Operation:**

Rt = Ra \* Rb or Rt = Ra \* Imm

**Instruction Formats:**

**MULS Rt, Ra, Rb [, Rp] – Register direct**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 31 29 | 28 24 | 23 | 22 18 | 17 | 16 12 | 11 7 | 6 5 | 4 0 |
| Ct3 | Rp5 | 0 | Rb5 | 01 | Ra5 | Rt5 | Sz2 | 65 |

**MULS Rt,Ra,Imm10**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 31 29 | 28 24 | 23 | 22 18 | 17 | 16 12 | 11 7 | 6 5 | 4 0 |
| Ct3 | Imm9..5 | 0 | Imm4..0 | 11 | Ra5 | Rt5 | Sz2 | 65 |

**MULS Rt,Ra,Imm32**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 31 29 | 28 24 | 23 | 22 18 | 17 | 16 12 | 11 7 | 6 5 | 4 0 |
| Ct3 | 165 | 0 | ~5 | 11 | Ra5 | Rt5 | Sz2 | 65 |
| Immediate32 | | | | | | | | |

**MULS Rt,Ra,Imm64**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 31 29 | 28 24 | 23 | 22 18 | 17 | 16 12 | 11 7 | 6 5 | 4 0 |
| Ct3 | 175 | 0 | ~5 | 11 | Ra5 | Rt5 | Sz2 | 65 |
| Immediate31..0 | | | | | | | | |
| Immediate63..32 | | | | | | | | |

**Clock Cycles:** 8

**Execution Units:** All Integer ALU’s

**Exceptions:** none

**Notes:**

### MULU – Unsigned Multiplication

**Description:**

Multiply two source operands and place the sum in the target register. All registers are integer registers. Arithmetic is signed twos-complement values. The ‘S’ flag indicates to perform an unsigned multiply. Unsigned multiply can be used during index calculations.

The high order product bits may be placed in register Rp for the register direct form of the instruction.

**Supported Operand Sizes:** .b, .w, .t, .o

**Flag Updates:**

Condition Register Ct3 is always updated.

If the result is zero the zero flag is set otherwise it is cleared.

If the result is negative the negative flag is set, otherwise it is cleared.

If there is a carry the carry flag is set in the condition register, otherwise it is cleared.

If there is an overflow, the overflow flag is set in the condition register, otherwise it is cleared.

Overflow for an unsigned multiply is detected when the product bits 64 to 127 are non-zero. For signed multiply overflow is detected as product bits 64 to 127 not equalling bit 63, the sign bit.

**Operation:**

Rt = Ra \* Rb or Rt = Ra \* Imm

**Instruction Formats:**

**MULU Rt, Ra, Rb [, Rp] – Register direct**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 31 29 | 28 24 | 23 | 22 18 | 17 | 16 12 | 11 7 | 6 5 | 4 0 |
| Ct3 | Rp5 | 1 | Rb5 | 01 | Ra5 | Rt5 | Sz2 | 65 |

**MULU Rt,Ra,Imm10**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 31 29 | 28 24 | 23 | 22 18 | 17 | 16 12 | 11 7 | 6 5 | 4 0 |
| Ct3 | Imm9..5 | 1 | Imm4..0 | 11 | Ra5 | Rt5 | Sz2 | 65 |

**MULU Rt,Ra,Imm32**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 31 29 | 28 24 | 23 | 22 18 | 17 | 16 12 | 11 7 | 6 5 | 4 0 |
| Ct3 | 165 | 1 | ~5 | 11 | Ra5 | Rt5 | Sz2 | 65 |
| Immediate32 | | | | | | | | |

**MULU Rt,Ra,Imm64**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 31 29 | 28 24 | 23 | 22 18 | 17 | 16 12 | 11 7 | 6 5 | 4 0 |
| Ct3 | 175 | 1 | ~5 | 11 | Ra5 | Rt5 | Sz2 | 65 |
| Immediate31..0 | | | | | | | | |
| Immediate63..32 | | | | | | | | |

**Clock Cycles:** 8

**Execution Units:** All Integer ALU’s

**Exceptions:** none

**Notes:**

### OR – Bitwise Or

**Description:**

Bitwise ‘or’ two source operands and place the sum in the target register. All registers are integer registers. Arithmetic is signed twos-complement values.

**Supported Operand Sizes:** .b, .w, .t, .o

**Flag Updates:**

Condition Register Ct3 is always updated.

If the result is zero the equals flag is set otherwise it is cleared.

If the result is negative the signed less than flag is set, otherwise it is cleared.

**Operation:**

Rt = Ra | Rb or Rt = Ra | Imm

**Instruction Formats:**

**OR Rt, Ra, Rb – Register direct**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 31 29 | 28 24 | 23 | 22 18 | 17 | 16 12 | 11 7 | 6 5 | 4 0 |
| Ct3 | ~4 | 0 | Rb5 | 01 | Ra5 | Rt5 | Sz2 | 95 |

**OR Rt,Ra,Imm10**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 31 29 | 28 24 | 23 | 22 18 | 17 | 16 12 | 11 7 | 6 5 | 4 0 |
| Ct3 | Imm9..5 | 0 | Imm4..0 | 11 | Ra5 | Rt5 | Sz2 | 95 |

**OR Rt,Ra,Imm32**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 31 29 | 28 24 | 23 | 22 18 | 17 | 16 12 | 11 7 | 6 5 | 4 0 |
| Ct3 | 165 | 0 | ~5 | 11 | Ra5 | Rt5 | Sz2 | 95 |
| Immediate32 | | | | | | | | |

**OR Rt,Ra,Imm64**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 31 29 | 28 24 | 23 | 22 18 | 17 | 16 12 | 11 7 | 6 5 | 4 0 |
| Ct3 | 175 | 0 | ~5 | 11 | Ra5 | Rt5 | Sz2 | 95 |
| Immediate31..0 | | | | | | | | |
| Immediate63..32 | | | | | | | | |

**Clock Cycles:** 2

**Execution Units:** All Integer ALU’s

**Exceptions:** none

**Notes:**

### SUB - Subtraction

**Description:**

Subtract two source operands and place the sum in the target register. All registers are integer registers. Arithmetic is signed twos-complement values.

**Supported Operand Sizes:** .b, .w, .t, .o

**Operation:**

Rt = Ra - Rb or Rt = Ra – Imm or Rt = Imm - Ra

**Clock Cycles:**

**Execution Units:** All Integer ALU’s

**Exceptions:** none

**Notes:**

**Instruction Formats:**

**SUB Rt, Ra, Rb – Register direct**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 31 29 | 28 24 | 23 | 22 18 | 17 | 16 12 | 11 7 | 6 5 | 4 0 |
| Ct3 | ~5 | 0 | Rb5 | 01 | Ra5 | Rt5 | Sz2 | 55 |

**Clock Cycles: 4**

**SUB Rt,Ra,Imm10**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 31 29 | 28 24 | 23 | 22 18 | 17 | 16 12 | 11 7 | 6 5 | 4 0 |
| Ct3 | Imm9..5 | S | Imm4..0 | 11 | Ra5 | Rt5 | Sz2 | 55 |

**Clock Cycles: 4**

**SUB Rt,Ra,Imm32**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 31 29 | 28 24 | 23 | 22 18 | 17 | 16 12 | 11 7 | 6 5 | 4 0 |
| Ct3 | 165 | S | ~5 | 11 | Ra5 | Rt5 | Sz2 | 55 |
| Immediate32 | | | | | | | | |

**Clock Cycles: 6**

**SUB Rt,Ra,Imm64**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 31 29 | 28 24 | 23 | 22 18 | 17 | 16 12 | 11 7 | 6 5 | 4 0 |
| Ct3 | 175 | S | ~5 | 11 | Ra5 | Rt5 | Sz2 | 55 |
| Immediate31..0 | | | | | | | | |
| Immediate63..32 | | | | | | | | |

**Clock Cycles: 8**

## Floating-Point Operations

### FCMP - Comparison

**Description:**

Compare two source operands and place the result in the target condition register. The result is a vector identifying the relationship between the two source operands as floating-point values.

**Supported Operand Sizes:** .h, .w, .t, .o

**Condition Register Format:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 3 | 2 | 1 | 0 |
| Integer | nf | zf | vf | cf |
| Float | lt | eq | ord | ~ |

**Related Branch Condition:**

|  |  |  |
| --- | --- | --- |
| **Cond5** | **Mnemonic** | **Test** |
| 0 | RA | 1 |
| 1 | SR | 0 |
| 2 | HI | !cf & !zf |
| 3 | LS | cf | zf |
| 4 | CC / HS | !cf |
| 5 | CS / LO | cf |
| 6 | NE | !zf |
| 7 | EQ | zf |
| 8 | VC | !vf |
| 9 | VS | vf |
| A | PL | !nf |
| B | MI | nf |
| C | GE | (nf & vf) | (!nf & !vf) |
| D | LT | (nf & !vf) | (!nf & vf) |
| E | GT | (nf & vf & !zf) | (!nf & !vf & zf) |
| F | LE | zf | (nf & !vf) | (!nf & vf) |
| 10 | FOR | ord |
| 11 | FUN | !ord |
| 12 | FGT | !(lt | eq) |
| 13 | FLE | lt | eq |
| 14 | FGE | !lt |
| 15 | FLT | lt |
| 16 | FNE | !eq |
| 17 | FEQ | eq & ord |
| 18 | - |  |
| 19 | - |  |
| 1A | - |  |
| 1B | - |  |
| 1C | - |  |
| 1D | - |  |
| 1E | - |  |
| 1F | - |  |

**Operation:**

Ct = Ra ? Rb or Ct = Ra ? Imm or Ct = Imm ? Ra

**Clock Cycles:**

**Execution Units:** All Integer ALU’s

**Exceptions:** none

**Notes:**

**Instruction Formats:**

**FCMP Ct, Ra, Rb – Register direct**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 31 29 | 28 24 | 23 | 22 18 | 17 | 16 12 | 11 8 | 7 | 6 5 | 4 0 |
| Ct3 | ~5 | 0 | Rb5 | 01 | Ra5 | ~4 | 1 | Sz2 | 35 |

**FCMP Ct,Ra,Imm10**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 31 29 | 28 24 | 23 | 22 18 | 17 | 16 12 | 11 8 | 7 | 6 5 | 4 0 |
| Ct3 | Imm9..5 | S | Imm4..0 | 11 | Ra5 | ~4 | 1 | Sz2 | 35 |

**FCMP Ct,Ra,Imm32**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 31 29 | 28 24 | 23 | 22 18 | 17 | 16 12 | 11 8 | 7 | 6 5 | 4 0 |
| Ct3 | 165 | S | ~5 | 11 | Ra5 | ~4 | 1 | Sz2 | 35 |
| Immediate32 | | | | | | | | | |

**FCMP Ct,Ra,Imm64**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 31 29 | 28 24 | 23 | 22 18 | 17 | 16 12 | 11 8 | 7 | 6 5 | 4 0 |
| Ct3 | 175 | S | ~5 | 11 | Ra5 | ~4 | 1 | Sz2 | 35 |
| Immediate31..0 | | | | | | | | | |
| Immediate63..32 | | | | | | | | | |

## Bit Manipulation Operations

### BCLR – Clear Bit

**Description:**

A bit in the source operand is cleared and the result placed in the target register. The previous bit status is stored in the zero flag of the condition register. The specified bit to clear is modulo the operand size.

**Supported Operand Sizes:** .b, .w, .t, .o

**Flag Updates:**

Condition Register Ct3 is always updated.

If the bit is zero the zero flag is set otherwise it is cleared.

**Operation:**

zf = Ra[Rb] or zf = Ra[Imm]

Rt = Ra &~bit Rb or Ra = Ra &~bit imm

**Instruction Formats:**

**BCLR Ct, Rt, Ra, Rb – Register direct**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 31 29 | 28 26 | 2524 | 23 | 22 18 | 17 | 16 12 | 11 7 | 6 5 | 4 0 |
| Ct3 | 03 | ~2 | 0 | Rb5 | 01 | Ra5 | Rt5 | Sz2 | 135 |

**Clock Cycles: 4**

**BCLR Ct, Rt, Ra,Imm7**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 31 29 | 28 26 | 25 24 | 23 | 22 18 | 17 | 16 12 | 11 7 | 6 5 | 4 0 |
| Ct3 | 03 | Im6..5 | 0 | Imm4..0 | 11 | Ra5 | Rt5 | Sz2 | 135 |

**Clock Cycles: 4**

**Clock Cycles:**

**Execution Units:** All Integer ALU’s

**Exceptions:** none

**Notes:**

### BCHG – Change Bit

**Description:**

A bit in the source operand is changed and placed in the target register. The previous bit status is stored in the zero flag of the condition register. The specified bit to change is modulo the operand size.

**Supported Operand Sizes:** .b, .w, .t, .o

**Flag Updates:**

Condition Register Ct3 is always updated.

If the bit is zero the zero flag is set otherwise it is cleared.

**Operation:**

zf = Ra[Rb] or zf = Ra[Imm]

Rt[Rb] = ~Ra[Rb] or Rt[Imm] = ~Ra[Imm]

**Instruction Formats:**

**BCLR Ct, Rt, Ra, Rb – Register direct**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 31 29 | 28 26 | 2524 | 23 | 22 18 | 17 | 16 12 | 11 7 | 6 5 | 4 0 |
| Ct3 | 23 | ~2 | 0 | Rb5 | 01 | Ra5 | Rt5 | Sz2 | 135 |

**Clock Cycles: 4**

**BCLR Ct, Rt, Ra,Imm7**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 31 29 | 28 26 | 25 24 | 23 | 22 18 | 17 | 16 12 | 11 7 | 6 5 | 4 0 |
| Ct3 | 23 | Im6..5 | 0 | Imm4..0 | 11 | Ra5 | Rt5 | Sz2 | 135 |

**Clock Cycles: 4**

**Clock Cycles:**

**Execution Units:** All Integer ALU’s

**Exceptions:** none

**Notes:**

### BPCHG – Change Bit Pair

**Description:**

A bit pair in the source operand is changed and placed in the target register. The pair is exclusively or’d with 11b. The previous bit status is stored in the flags of the condition register. There are four bit-pairs per byte indicated as pair #0 to #3. The bit pair specified is taken modulo the operand size.

|  |  |
| --- | --- |
| Bit Pair Value | Updated Value |
| 00 | 11 |
| 01 | 10 |
| 10 | 01 |
| 11 | 00 |

**Supported Operand Sizes:** .b, .w, .t, .o

**Flag Updates:**

Condition Register Ct3 is always updated.

|  |  |
| --- | --- |
| Bit Pair Value | Flag Setting |
| 00 | Zero flag is set, cf, nf, and vf are cleared |
| 01 | Carry flag is set, zf, nf, and vf are cleared |
| 10 | Negative flag is set, zf, cf, and vf are cleared |
| 11 | Overflow flags is set, nf, zf and cf are cleared |

**Operation:**

zf = Ra[Rb] or zf = Ra[Imm]

Rt[Rb] = ~Ra[Rb] or Rt[Imm] = ~Ra[Imm]

**Instruction Formats:**

**BCLR Ct, Rt, Ra, Rb – Register direct**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 31 29 | 28 26 | 2524 | 23 | 22 18 | 17 | 16 12 | 11 7 | 6 5 | 4 0 |
| Ct3 | 63 | ~2 | 0 | Rb5 | 01 | Ra5 | Rt5 | Sz2 | 135 |

**Clock Cycles: 4**

**BCLR Ct, Rt, Ra,Imm7**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 31 29 | 28 26 | 25 24 | 23 | 22 18 | 17 | 16 12 | 11 7 | 6 5 | 4 0 |
| Ct3 | 63 | Im6..5 | 0 | Imm4..0 | 11 | Ra5 | Rt5 | Sz2 | 135 |

**Clock Cycles: 4**

**Clock Cycles:**

**Execution Units:** All Integer ALU’s

**Exceptions:** none

**Notes:**

### BPCLR – Clear Bit Pair

**Description:**

A pair of bits in the source operand is cleared and the result placed in the target register. The previous bit status is stored in the flags of the condition register.

**Supported Operand Sizes:** .b, .w, .t, .o

**Flag Updates:**

Condition Register Ct3 is always updated.

|  |  |
| --- | --- |
| Bit Pair Value | Flag Setting |
| 00 | Zero flag is set, cf, nf, and vf are cleared |
| 01 | Carry flag is set, zf, nf, and vf are cleared |
| 10 | Negative flag is set, zf, cf, and vf are cleared |
| 11 | Overflow flags is set, nf, zf and cf are cleared |

**Operation:**

zf = Ra[Rb] or zf = Ra[Imm]

Rt = Ra &~bit Rb or Ra = Ra &~bit imm

**Instruction Formats:**

**BPCLR Ct, Rt, Ra, Rb – Register direct**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 31 29 | 28 26 | 2524 | 23 | 22 18 | 17 | 16 12 | 11 7 | 6 5 | 4 0 |
| Ct3 | 43 | ~2 | 0 | Rb5 | 01 | Ra5 | Rt5 | Sz2 | 135 |

**Clock Cycles: 4**

**BPCLR Ct, Rt, Ra,Imm7**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 31 29 | 28 26 | 25 24 | 23 | 22 18 | 17 | 16 12 | 11 7 | 6 5 | 4 0 |
| Ct3 | 43 | Im6..5 | 0 | Imm4..0 | 11 | Ra5 | Rt5 | Sz2 | 135 |

**Clock Cycles: 4**

**Clock Cycles:**

**Execution Units:** All Integer ALU’s

**Exceptions:** none

**Notes:**

### BPTST – Test Bit Pair

**Description:**

Test a bit pair in the source operand and place the bit status in the flags of the condition register. The bit tested is modulo the operation size. All combinations of bit pair value may be detected via flags.

**Supported Operand Sizes:** .b, .w, .t, .o

**Flag Updates:**

Condition Register Ct3 is always updated.

|  |  |
| --- | --- |
| Bit Pair Value | Flag Setting |
| 00 | Zero flag is set, cf, nf, and vf are cleared |
| 01 | Carry flag is set, zf, nf, and vf are cleared |
| 10 | Negative flag is set, zf, cf, and vf are cleared |
| 11 | Overflow flags is set, nf, zf and cf are cleared |

If the bit pair is zero the zero flag is set otherwise it is cleared.

If the bit pair is

The negative flag is set to the value of the high order bit of the pair.

The overflow flag is set to the exclusive or of the two bits in the pair.

**Operation:**

**Instruction Formats:**

**BPTST Ct, Ra, Rb – Register direct**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 31 29 | 28 26 | 2524 | 23 | 22 18 | 17 | 16 12 | 11 7 | 6 5 | 4 0 |
| Ct3 | 73 | ~2 | 0 | Rb5 | 01 | Ra5 | ~5 | Sz2 | 135 |

**Clock Cycles: 4**

**BPTST Ct,Ra,Imm7**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 31 29 | 28 26 | 25 24 | 23 | 22 18 | 17 | 16 12 | 11 7 | 6 5 | 4 0 |
| Ct3 | 73 | Im6..5 | 0 | Imm4..0 | 11 | Ra5 | ~5 | Sz2 | 135 |

**Clock Cycles: 4**

**Clock Cycles:**

**Execution Units:** All Integer ALU’s

**Exceptions:** none

**Notes:**

### BSET – Set Bit

**Description:**

A bit in the source operand is set and placed in the target register. The previous bit status is stored in the zero flag of the condition register.

**Supported Operand Sizes:** .b, .w, .t, .o

**Flag Updates:**

Condition Register Ct3 is always updated.

If the bit is zero the zero flag is set otherwise it is cleared.

**Operation:**

zf = Ra[Rb] or zf = Ra[Imm]

Rt = Ra | bit Rb or Rt = Ra or Bit[Imm]

**Instruction Formats:**

**BCLR Ct, Rt, Ra, Rb – Register direct**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 31 29 | 28 26 | 2524 | 23 | 22 18 | 17 | 16 12 | 11 7 | 6 5 | 4 0 |
| Ct3 | 13 | ~2 | 0 | Rb5 | 01 | Ra5 | Rt5 | Sz2 | 135 |

**Clock Cycles: 4**

**BCLR Ct, Rt, Ra,Imm7**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 31 29 | 28 26 | 25 24 | 23 | 22 18 | 17 | 16 12 | 11 7 | 6 5 | 4 0 |
| Ct3 | 13 | Im6..5 | 0 | Imm4..0 | 11 | Ra5 | Rt5 | Sz2 | 135 |

**Clock Cycles: 4**

**Clock Cycles:**

**Execution Units:** All Integer ALU’s

**Exceptions:** none

**Notes:**

### BTST – Test Bit

**Description:**

Test a bit in the source operand and place the bit status in the zero flag of the condition register. The bit tested is modulo the operation size.

**Supported Operand Sizes:** .b, .w, .t, .o

**Flag Updates:**

Condition Register Ct3 is always updated.

If the bit is zero the zero flag is set otherwise it is cleared.

**Operation:**

zf = Ra[Rb] or zf = Ra[Imm]

**Instruction Formats:**

**BTST Ct, Ra, Rb – Register direct**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 31 29 | 28 26 | 2524 | 23 | 22 18 | 17 | 16 12 | 11 7 | 6 5 | 4 0 |
| Ct3 | 33 | ~2 | 0 | Rb5 | 01 | Ra5 | ~5 | Sz2 | 135 |

**Clock Cycles: 4**

**BTST Ct,Ra,Imm7**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 31 29 | 28 26 | 25 24 | 23 | 22 18 | 17 | 16 12 | 11 7 | 6 5 | 4 0 |
| Ct3 | 33 | Im6..5 | 0 | Imm4..0 | 11 | Ra5 | ~5 | Sz2 | 135 |

**Clock Cycles: 4**

**Clock Cycles:**

**Execution Units:** All Integer ALU’s

**Exceptions:** none

**Notes:**

## Shift and Rotate Operations

### ASL – Arithmetic Shift Left

**Description:**

Shift the first source operand to the left by the number of bits specified by the second source operand and place the result in the target register. All registers are integer registers. Arithmetic is signed twos-complement values. The least significant bit is filled with the value of ‘N’ specified in the instruction.

**Supported Operand Sizes:** .b, .w, .t, .o

**Flag Updates:**

Condition Register Ct3 is always updated.

If the result is zero the equals flag is set otherwise it is cleared.

If the result is negative the signed less than flag is set, otherwise it is cleared.

If the sign bit changes at any stage during the shift, the overflow flag is set, otherwise it is cleared.

The carry flag is set to the last bit shifted out of the most significant bit of the operand.

**Operation:**

Rt = Ra << Rb or Rt = Ra << Imm

**Instruction Formats:**

**ASL Rt, Ra, Rb – Register direct**

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 31 29 | 28 27 | 26 | 25 | 24 | 23 | 22 18 | 17 | 16 12 | 11 7 | 6 5 | 4 0 |
| Ct3 | 02 | ~ | N |  | 0 | Rb5 | 01 | Ra5 | Rt5 | Sz2 | 145 |

**Clock Cycles: 4 + shift amount**

**ASL Rt,Ra,Imm6**

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 31 29 | 28 27 | 26 | 25 | 24 | 23 | 22 18 | 17 | 16 12 | 11 7 | 6 5 | 4 0 |
| Ct3 | 02 | ~ | N | I | 0 | Imm4..0 | 11 | Ra5 | Rt5 | Sz2 | 145 |

**Clock Cycles: 4 + shift amount**

**Clock Cycles:**

**Execution Units:** All Integer ALU’s

**Exceptions:** none

**Notes:**

### ASR – Arithmetic Shift Right

**Description:**

Shift the first source operand to the right, preserving the sign bit, by the number of bits specified by the second source operand and place the result in the target register. All registers are integer registers. Arithmetic is signed twos-complement values.

**Supported Operand Sizes:** .b, .w, .t, .o

**Flag Updates:**

Condition Register Ct3 is always updated.

If the result is zero the equals flag is set otherwise it is cleared.

If the result is negative the signed less than flag is set, otherwise it is cleared.

The overflow flag is cleared.

The carry flag is set to the last bit shifted out of the least significant bit of the operand.

**Operation:**

Rt = Ra >> Rb or Rt = Ra >> Imm

**Instruction Formats:**

**ASL Rt, Ra, Rb – Register direct**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 31 29 | 28 27 | 26 24 | 23 | 22 18 | 17 | 16 12 | 11 7 | 6 5 | 4 0 |
| Ct3 | 02 | ~3 | 1 | Rb5 | 01 | Ra5 | Rt5 | Sz2 | 145 |

**Clock Cycles: 4 + shift amount**

**ASL Rt,Ra,Imm6**

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 31 29 | 28 27 | 2625 | 24 | 23 | 22 18 | 17 | 16 12 | 11 7 | 6 5 | 4 0 |
| Ct3 | 02 | ~2 | I | 1 | Imm4..0 | 11 | Ra5 | Rt5 | Sz2 | 145 |

**Clock Cycles: 4 + shift amount**

**Clock Cycles:**

**Execution Units:** All Integer ALU’s

**Exceptions:** none

**Notes:**

### LSL – Logical Shift Left

**Description:**

Shift the first source operand to the left by the number of bits specified by the second source operand and place the result in the target register. All registers are integer registers. Arithmetic is signed twos-complement values. Fill the least significant bit with the value specified by ‘N’ in the instruction.

**Supported Operand Sizes:** .b, .w, .t, .o

**Flag Updates:**

Condition Register Ct3 is always updated.

If the result is zero the equals flag is set otherwise it is cleared.

If the result is negative the signed less than flag is set, otherwise it is cleared.

The overflow flag is cleared.

The carry flag is set to the last bit shifted out of the most significant bit of the operand.

**Operation:**

Rt = Ra << Rb or Rt = Ra << Imm

**Instruction Formats:**

**LSL Rt, Ra, Rb – Register direct**

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 31 29 | 28 27 | 26 | 25 | 24 | 23 | 22 18 | 17 | 16 12 | 11 7 | 6 5 | 4 0 |
| Ct3 | 12 | ~ | N | ~ | 0 | Rb5 | 01 | Ra5 | Rt5 | Sz2 | 145 |

**LSL Rt,Ra,Imm6**

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 31 29 | 28 27 | 26 | 25 | 24 | 23 | 22 18 | 17 | 16 12 | 11 7 | 6 5 | 4 0 |
| Ct3 | 12 | ~ | N | I | 0 | Imm4..0 | 11 | Ra5 | Rt5 | Sz2 | 145 |

**Clock Cycles:**

**Execution Units:** All Integer ALU’s

**Exceptions:** none

**Notes:**

### LSR – Logical Shift Right

**Description:**

Shift the first source operand to the right by the number of bits specified by the second source operand and place the result in the target register. All registers are integer registers. Arithmetic is signed twos-complement values. Fill the least significant bit with the value specified by ‘N’ in the instruction.

**Supported Operand Sizes:** .b, .w, .t, .o

**Flag Updates:**

Condition Register Ct3 is always updated.

If the result is zero the equals flag is set otherwise it is cleared.

If the result is negative the signed less than flag is set, otherwise it is cleared.

The overflow flag is cleared.

The carry flag is set to the last bit shifted out of the least significant bit of the operand.

**Operation:**

Rt = Ra >> Rb or Rt = Ra >> Imm

**Instruction Formats:**

**LSR Rt, Ra, Rb – Register direct**

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 31 29 | 28 27 | 26 | 25 | 24 | 23 | 22 18 | 17 | 16 12 | 11 7 | 6 5 | 4 0 |
| Ct3 | 12 | ~ | N | ~ | 1 | Rb5 | 01 | Ra5 | Rt5 | Sz2 | 145 |

**LSR Rt,Ra,Imm6**

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 31 29 | 28 27 | 26 | 25 | 24 | 23 | 22 18 | 17 | 16 12 | 11 7 | 6 5 | 4 0 |
| Ct3 | 12 | ~ | N | I | 1 | Imm4..0 | 11 | Ra5 | Rt5 | Sz2 | 145 |

**Clock Cycles:**

**Execution Units:** All Integer ALU’s

**Exceptions:** none

**Notes:**

### ROL – Rotate Left

**Description:**

Rotate the first source operand to the left by the number of bits specified by the second source operand and place the result in the target register. All registers are integer registers. Arithmetic is signed twos-complement values. The least significant bit is set to the value of the most significant bit exclusively or’d with the value ‘N’ from the instruction.

**Supported Operand Sizes:** .b, .w, .t, .o

**Flag Updates:**

Condition Register Ct3 is always updated.

If the result is zero the equals flag is set otherwise it is cleared.

If the result is negative the signed less than flag is set, otherwise it is cleared.

The overflow flag is cleared.

The carry flag is set to the last bit shifted out of the most significant bit of the operand.

**Operation:**

Rt = Ra << Rb or Rt = Ra << Imm

**Instruction Formats:**

**ROL Rt, Ra, Rb – Register direct**

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 31 29 | 28 27 | 26 | 25 | 24 | 23 | 22 18 | 17 | 16 12 | 11 7 | 6 5 | 4 0 |
| Ct3 | 22 | ~ | N | ~ | 0 | Rb5 | 01 | Ra5 | Rt5 | Sz2 | 145 |

**ROL Rt,Ra,Imm6**

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 31 29 | 28 27 | 26 | 25 | 24 | 23 | 22 18 | 17 | 16 12 | 11 7 | 6 5 | 4 0 |
| Ct3 | 22 | ~ | N | I | 0 | Imm4..0 | 11 | Ra5 | Rt5 | Sz2 | 145 |

**Clock Cycles:**

**Execution Units:** All Integer ALU’s

**Exceptions:** none

**Notes:**

### ROLC – Rotate Left through Carry

**Description:**

Rotate the first source operand through the carry to the left by the number of bits specified by the second source operand and place the result in the target register. All registers are integer registers. Arithmetic is signed twos-complement values. The least significant bit is set to the value of the carry bit exclusively or’d with the value ‘N’ from the instruction.

**Supported Operand Sizes:** .b, .w, .t, .o

**Flag Updates:**

Condition Register Ct3 is always updated.

If the result is zero the equals flag is set otherwise it is cleared.

If the result is negative the signed less than flag is set, otherwise it is cleared.

The overflow flag is cleared.

The carry flag is set to the exclusive or of the last bit shifted out of the most significant bit of the operand and the value of ‘N’ in the instruction.

**Operation:**

Rt = Ra << Rb or Rt = Ra << Imm

**Instruction Formats:**

**ROLC Rt, Ra, Rb – Register direct**

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 31 29 | 28 27 | 26 | 25 | 24 | 23 | 22 18 | 17 | 16 12 | 11 7 | 6 5 | 4 0 |
| Ct3 | 32 | ~ | N | ~ | 0 | Rb5 | 01 | Ra5 | Rt5 | Sz2 | 145 |

**ROLC Rt,Ra,Imm6**

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 31 29 | 28 27 | 26 | 25 | 24 | 23 | 22 18 | 17 | 16 12 | 11 7 | 6 5 | 4 0 |
| Ct3 | 32 | ~ | N | I | 0 | Imm4..0 | 11 | Ra5 | Rt5 | Sz2 | 145 |

**Clock Cycles:**

**Execution Units:** All Integer ALU’s

**Exceptions:** none

**Notes:**

### ROR – Rotate Right

**Description:**

Rotate the first source operand through the carry to the right by the number of bits specified by the second source operand and place the result in the target register. All registers are integer registers. Arithmetic is signed twos-complement values. The most significant bit is set to the value of the least significant bit exclusively or’d with the value ‘N’ from the instruction.

**Supported Operand Sizes:** .b, .w, .t, .o

**Flag Updates:**

Condition Register Ct3 is always updated.

If the result is zero the equals flag is set otherwise it is cleared.

If the result is negative the signed less than flag is set, otherwise it is cleared.

The overflow flag is cleared.

The carry flag is set to the last bit shifted out of the least significant bit of the operand.

**Operation:**

Rt = Ra >> Rb or Rt = Ra >> Imm

**Instruction Formats:**

**ROR Rt, Ra, Rb – Register direct**

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 31 29 | 28 27 | 26 | 25 | 24 | 23 | 22 18 | 17 | 16 12 | 11 7 | 6 5 | 4 0 |
| Ct3 | 22 | ~ | N | ~ | 1 | Rb5 | 01 | Ra5 | Rt5 | Sz2 | 145 |

**ROR Rt,Ra,Imm6**

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 31 29 | 28 27 | 26 | 25 | 24 | 23 | 22 18 | 17 | 16 12 | 11 7 | 6 5 | 4 0 |
| Ct3 | 22 | ~ | N | I | 1 | Imm4..0 | 11 | Ra5 | Rt5 | Sz2 | 145 |

**Clock Cycles:**

**Execution Units:** All Integer ALU’s

**Exceptions:** none

**Notes:**

### RORC – Rotate Right through Carry

**Description:**

Rotate the first source operand through the carry to the right by the number of bits specified by the second source operand and place the result in the target register. All registers are integer registers. Arithmetic is signed twos-complement values. The most significant bit is set to the value of the carry bit exclusively or’d with the value ‘N’ from the instruction.

**Supported Operand Sizes:** .b, .w, .t, .o

**Flag Updates:**

Condition Register Ct3 is always updated.

If the result is zero the equals flag is set otherwise it is cleared.

If the result is negative the signed less than flag is set, otherwise it is cleared.

The overflow flag is cleared.

The carry flag is set to the last bit shifted out of the least significant bit of the operand.

**Operation:**

Rt = Ra >> Rb or Rt = Ra >> Imm

**Instruction Formats:**

**RORC Rt, Ra, Rb – Register direct**

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 31 29 | 28 27 | 26 | 25 | 24 | 23 | 22 18 | 17 | 16 12 | 11 7 | 6 5 | 4 0 |
| Ct3 | 32 | ~ | N | ~ | 1 | Rb5 | 01 | Ra5 | Rt5 | Sz2 | 145 |

**RORC Rt,Ra,Imm6**

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 31 29 | 28 27 | 26 | 25 | 24 | 23 | 22 18 | 17 | 16 12 | 11 7 | 6 5 | 4 0 |
| Ct3 | 32 | ~ | N | I | 1 | Imm4..0 | 11 | Ra5 | Rt5 | Sz2 | 145 |

**Clock Cycles:**

**Execution Units:** All Integer ALU’s

**Exceptions:** none

**Notes:**

### SWAP – Swap register halves

**Description:**

SWAP.T: The bits 0 to 31 and 32 to 63 are swapped.

SWAP.W: The bits 0 to 15 and 16 to 31 are swapped.

SWAP.B: The bits 0 to 7 and 8 to 15 are swapped.

**Supported Operand Sizes:** .b, .w, .t

**Instruction Format:**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 31 29 | 28 24 | 23 | 22 18 | 17 | 16 12 | 11 7 | 6 5 | 4 0 |
| Ct3 | ~5 | 0 | 165 | 01 | Ra5 | Rt5 | Sz2 | 15 |

## Flow Control Instructions

### ATRAP – Application Trap

**Description:**

Execute trap. The data field is loaded into the specified target register, Rt. The trap number to execute comes from the contents of register Ra or an immediate value encoded in the instruction. There are 32 vectors reserved for application use. See the section on exceptions.

**Instruction Format:**

**ATRAP Rt, Ra, #Data**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 31 29 | 28 18 | 17 | 16 12 | 11 7 | 6 5 | 4 0 |
| ~3 | Data10..0 | 01 | Ra5 | Rt5 | 32 | 05 |

**ATRAP Rt, #Num, #Data**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 31 29 | 28 18 | 17 | 16 12 | 11 7 | 6 5 | 4 0 |
| ~3 | Data10..0 | 11 | Num5 | Rt5 | 32 | 05 |

**Operation:**

The program counter and condition code register group are pushed on the stack. Next the vector is fetched from the application’s exception vector table and jumped to. The processor remains in user or application mode.

### Bcc – Conditional Branch

Bcc Cn, label

**Description:**

Branch if the condition is met. The BSR instruction places the address of the next instruction on the stack. The displacement is relative to the address of the branch instruction. The branch range is +/- 1MB.

**Instruction Format:**

|  |  |  |  |
| --- | --- | --- | --- |
| 31 13 | 12 10 | 9 5 | 4 0 |
| Disp20..2 | Cr3 | Cond5 | 285 |

|  |  |  |
| --- | --- | --- |
| **Cond5** | **Mnemonic** | **Test** |
| 0 | RA | 1 |
| 1 | SR | 0 |
| 2 | HI | !cf & !zf |
| 3 | LS | cf | zf |
| 4 | CC / HS | !cf |
| 5 | CS / LO | cf |
| 6 | NE | !zf |
| 7 | EQ | zf |
| 8 | VC | !vf |
| 9 | VS | vf |
| A | PL | !nf |
| B | MI | nf |
| C | GE | (nf & vf) | (!nf & !vf) |
| D | LT | (nf & !vf) | (!nf & vf) |
| E | GT | (nf & vf & !zf) | (!nf & !vf & zf) |
| F | LE | zf | (nf & !vf) | (!nf & vf) |
| 10 | FOR |  |
| 11 | FUN |  |
| 12 | FGT |  |
| 13 | FLE |  |
| 14 | FGE |  |
| 15 | FLT |  |
| 16 | FNE |  |
| 17 | FEQ |  |
| 18 | - |  |
| 19 | - |  |
| 1A | - |  |
| 1B | - |  |
| 1C | - |  |
| 1D | - |  |
| 1E | - |  |
| 1F | - |  |

**Clock Cycles: 4**

### BRA – Unconditional Branch

**Description:**

Unconditionally branch to a new program address. The displacement is relative to the address of the branch instruction. The branch range is +/- 8MB.

**Instruction Format:**

|  |  |  |  |
| --- | --- | --- | --- |
| 31 13 | 12 10 | 9 5 | 4 0 |
| Disp20..2 | D23..21 | 05 | 285 |

**Clock Cycles: 3**

### BRK – Breakpoint

**Description:**

Execute the breakpoint exception. This is a form of the TRAP instruction.

**Instruction Format:**

|  |  |
| --- | --- |
| 31 5 | 4 0 |
| 0 | 05 |

### BSR – Branch to Subroutine

**Description:**

Branch to a subroutine placing the address of the next instruction on the stack. The displacement is relative to the address of the branch instruction. The branch range is +/- 8MB.

**Instruction Format:**

|  |  |  |  |
| --- | --- | --- | --- |
| 31 13 | 12 10 | 9 5 | 4 0 |
| Disp20..2 | D23..21 | 15 | 285 |

**Clock Cycles: 8**

### DBcc – Decrement and Branch

DBcc Cn,label

**Description:**

Decrement the loop counter and branch if the condition is false and the loop counter is not equal to minus one. The displacement is relative to the address of the branch instruction. The branch range is +/- 1MB.

**Instruction Format:**

|  |  |  |  |
| --- | --- | --- | --- |
| 31 13 | 12 10 | 9 5 | 4 0 |
| Disp20..2 | Cr3 | Cond5 | 285 |

### JMP – Jump to Address

**Description:**

Compute the effective address and jump to it.

**Flag Updates:**

None.

**Operation:**

PC = Ra + Rb or PC = Ra + Imm

**Clock Cycles:**

**Execution Units:** All Integer ALU’s

**Exceptions:** none

**Notes:**

**Instruction Formats:**

**JMP (Ra, Rb) – Register direct**

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 31 | 30 | 29 | 28 24 | 23 | 22 18 | 17 | 16 12 | 11 7 | 6 5 | 4 0 |
| ~ | T | 0 | ~5 | 0 | Rb5 | 01 | Ra5 | ~5 | Sz2 | 245 |

**Clock Cycles: 7**

**JMP Imm10 (Ra)**

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 31 | 30 | 29 | 28 24 | 23 | 22 18 | 17 | 16 12 | 11 7 | 6 5 | 4 0 |
| ~ | T | 0 | Imm9..5 | 0 | Imm4..0 | 11 | Ra5 | ~5 | Sz2 | 245 |

**Clock Cycles: 7**

**JMP Imm32 (Ra)**

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 31 | 30 | 29 | 28 24 | 23 | 22 18 | 17 | 16 12 | 11 7 | 6 5 | 4 0 |
| ~ | T | 0 | 165 | 0 | ~5 | 11 | Ra5 | ~5 | Sz2 | 245 |
| Immediate32 | | | | | | | | | | |

**Clock Cycles: 10**

**JMP Imm64 (Ra)**

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 31 | 30 | 29 | 28 24 | 23 | 22 18 | 17 | 16 12 | 11 7 | 6 5 | 4 0 |
| ~ | T | 0 | 175 | 0 | ~5 | 11 | Ra5 | ~5 | Sz2 | 245 |
| Immediate31..0 | | | | | | | | | | |
| Immediate63..32 | | | | | | | | | | |

**Clock Cycles: 13**

### JSR – Jump to Subroutine

**Description:**

Compute the effective address and jump to it. The address of the next instruction is placed on the stack. If bit ‘G’ in the instruction is set then save the condition code register group, CCRG to the stack. If the ‘T’ bit of the instruction is set, then execute the call target trap exception if the instruction at the target address is not a call target instruction.

**Flag Updates:**

None.

**Operation:**

Memory[sp-8] = next PC

SP = SP - 8

If (G)

Memory[sp-8] = CCRG

SP = SP - 8

PC = Ra + Rb or PC = Ra + Imm

**Clock Cycles:**

**Execution Units:** All Integer ALU’s

**Exceptions:** none

**Notes:**

**Instruction Formats:**

**JSR (Ra, Rb) – Register direct**

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 31 | 30 | 29 | 28 24 | 23 | 22 18 | 17 | 16 12 | 11 7 | 6 5 | 4 0 |
| ~ | T | G | ~5 | 1 | Rb5 | 01 | Ra5 | ~5 | Sz2 | 245 |

**Clock Cycles: 7**

**JSR Imm10 (Ra)**

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 31 | 30 | 29 | 28 24 | 23 | 22 18 | 17 | 16 12 | 11 7 | 6 5 | 4 0 |
| ~ | T | G | Imm9..5 | 1 | Imm4..0 | 11 | Ra5 | ~5 | Sz2 | 245 |

**Clock Cycles: 7**

**JSR Imm32 (Ra)**

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 31 | 30 | 29 | 28 24 | 23 | 22 18 | 17 | 16 12 | 11 7 | 6 5 | 4 0 |
| ~ | T | G | 165 | 1 | ~5 | 11 | Ra5 | ~5 | Sz2 | 245 |
| Immediate32 | | | | | | | | | | |

**Clock Cycles: 10**

**JSR Imm64 (Ra)**

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 31 | 30 | 29 | 28 24 | 23 | 22 18 | 17 | 16 12 | 11 7 | 6 5 | 4 0 |
| ~ | T | G | 175 | 1 | ~5 | 11 | Ra5 | ~5 | Sz2 | 245 |
| Immediate31..0 | | | | | | | | | | |
| Immediate63..32 | | | | | | | | | | |

**Clock Cycles: 13**

### RTI – Return From Interrupt

**Instruction Formats:**

**RTI #Arg,#Rpt**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 31 29 | 28 24 | 23 | 22 18 | 17 | 16 12 | 11 7 | 6 5 | 4 0 |
| ~3 | 15 | 0 | Arg11..7 | 01 | Arg6..2 | Rpt5 | ~2 | 155 |

**Field Description:**

Rpt5 is the number of words to skip past the return address. This is to allow inline subroutine arguments. Up to 32 words may be skipped over. For externally triggered interrupts this field should be zero.

Arg11..2 is the number of words of subroutine arguments to remove from the stack. Up to 1024 words of arguments may be removed.

**Operation:**

Pop the status register, condition code group register, then the program counter from the stack. Add Rpt tetras to the program counter, and Arg tetras to the stack pointer.

### ARTI – Application Return from Interrupt

**Description:**

Restore the condition code register group from the stack and return to the calling routine. This instruction is used to return from a subroutine that was called which stored the CCRG on the stack. See [JSR](#_JSR_–_Jump) or [ATRAP](#_ATRAP_–_Application).

**Instruction Formats:**

**RTR #Arg,#Rpt**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 31 29 | 28 24 | 23 | 22 18 | 17 | 16 12 | 11 7 | 6 5 | 4 0 |
| ~3 | 25 | 0 | Arg11..7 | 01 | Arg6..2 | Rpt5 | ~2 | 155 |

**Field Description:**

Rpt5 is the number of words to skip past the return address. This is to allow inline subroutine arguments. Up to 32 words may be skipped over. For externally triggered interrupts this field should be zero.

Arg11..2 is the number of words of subroutine arguments to remove from the stack. Up to 1024 words of arguments may be removed.

**Operation:**

Pop the condition code group register then the program counter from the stack. Add Rpt tetras to the program counter, and Arg tetras to the stack pointer.

### RTS – Return from Subroutine

**Instruction Formats:**

**RTS #Arg,#Rpt**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 31 29 | 28 24 | 23 | 22 18 | 17 | 16 12 | 11 7 | 6 5 | 4 0 |
| ~3 | 05 | 0 | Arg11..7 | 01 | Arg6..2 | Rpt5 | ~2 | 155 |

**Field Description:**

Rpt5 is the number of tetras to skip past the return address. This is to allow inline subroutine arguments. Up to 32 tetras may be skipped over.

Arg11..2 is the number of tetras of subroutine arguments to remove from the stack. Up to 1024 tetras of arguments may be removed.

### TRAP – Trap

**Description:**

Execute trap. The data field is loaded into the specified target register, Rt. The trap number to execute comes from the contents of register Ra or an immediate value encoded in the instruction. The trap number must be between 1 and 255.

**Instruction Format:**

**TRAP Rt, Ra, #Data**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 31 29 | 28 18 | 17 | 16 12 | 11 7 | 6 5 | 4 0 |
| ~3 | Data10..0 | 01 | Ra5 | Rt5 | 02 | 05 |

**TRAP Rt, #Num, #Data**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 31 29 | 28 18 | 17 | 16 12 | 11 7 | 6 5 | 4 0 |
| Num7..5 | Data10..0 | 11 | Num4..0 | Rt5 | 02 | 05 |

**Operation:**

The program counter, condition code register group and the status register are pushed on the stack. Next the vector is fetched from the exception vector table and jumped to.

### TRAPV – Trap on Overflow

**Description:**

Execute trap if overflow occurred. The data field is loaded into the specified target register, Rt. The overflow bit in condition code register Cr3 is checked.

**Instruction Format:**

**TRAPV Rt, #Data**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 31 29 | 28 18 | 17 | 16 12 | 11 7 | 6 5 | 4 0 |
| Cr3 | Data10..0 | 11 | 315 | Rt5 | 22 | 05 |

**Operation:**

The program counter and condition code register group are pushed on the stack. Next the vector is fetched from the application’s exception vector table and jumped to. Note that this trap does not switch the processor’s operating mode.

## Memory Operations

### LOAD Rn,<sea>

**Description:**

Load register Rt from source.

**Supported Operand Sizes:** .b, .w, .t, .o

**Supported Address Modes:**

|  |  |  |  |
| --- | --- | --- | --- |
| 0 | 1 | 2 | 3 |
| D(Rn) | D(Rn)+ | +D(Rn) | D(Rn, Rn) |
| \* | \* | \* | \* |

**Instruction Formats: D(Rn), D(Rn)+, +D(Rn)**

If displacement bits 3 to 10 equal 80h then the displacement is 32 bits in the following instruction word. If displacement bits 2 to 10 equal 81h then the displacement is 64-bits in the following two instruction words. The value 82h in displacement bits 3 to 10 is reserved and should not be used.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 31 24 | 23 | 22 18 | 17 | 16 12 | 11 10 | 9 7 | 6 5 | 4 0 |
| Disp10..3 | C | Rb5 | 0 | Rt5 | Md2 | D2..0 | Sz2 | 165 |
| Opt Constant31..0 | | | | | | | | |
| Opt Constant63..32 | | | | | | | | |

**Instruction Format: D(Rn, Rn)**

IF the displacement D bits equal 32 then the next word is used as the displacement. If the displacement D bits equal 33 then the next two words are used as the displacement. A D displacement bits equal to 34 are reserved.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 31 29 | 28 24 | 23 | 22 18 | 17 | 16 12 | 11 10 | 9 7 | 6 5 | 4 0 |
| D5..3 | Rc5 | C | Rb5 | 0 | Rt5 | Md2 | D2..0 | Sz2 | 165 |
| Opt Constant31..0 | | | | | | | | | |
| Opt Constant63..32 | | | | | | | | | |

Notes:

### LOAD Reglist,<sea>

**Description:**

Load multiple registers from memory.

If bit ‘S’ of the instruction is set then the load addresses for registers not specified in the register list but does not load them. This allows values from memory to be skipped over. This could be used for loading thread state from a thread control block, omitting some of the registers from the load. Registers not specified are ‘skipped’ over, otherwise if ‘S’ is not set then registers specified are loaded from consecutive memory locations.

**Supported Operand Sizes:** .b, .w, .t, .o

**Supported Address Modes:**

|  |  |  |  |
| --- | --- | --- | --- |
| 0 | 1 | 2 | 3 |
| D(Rn) | D(Rn)+ | +D(Rn) | D(Rn, Rn) |
| \* | \* | \* | \* |

**Instruction Formats: D(Rn), D(Rn)+, +D(Rn)**

If displacement bits 3 to 10 equal 80h then the displacement is 32 bits in the following instruction word. If displacement bits 3 to 10 equal 81h then the displacement is 64-bits in the following two instruction words. The value 82h in displacement bits 3 to 10 is reserved and should not be used.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 31 24 | 23 | 22 18 | 17 | 16 12 | 11 9 | 8 7 | 6 5 | 4 0 |
| Disp10..3 | S | Rb5 | 0 | Special5 | Md2 | D2..0 | Sz2 | 175 |
| Reglist32 | | | | | | | | |
| Opt Constant31..0 | | | | | | | | |
| Opt Constant63..32 | | | | | | | | |

**Instruction Format: D(Rn, Rn)**

IF the displacement D bits equal 16 then the next word is used as the displacement. If the displacement D bits equal 17 then the next two words are used as the displacement. A D displacement bits equal to 18 are reserved.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 31 29 | 28 24 | 23 | 22 18 | 17 | 16 12 | 11 9 | 8 7 | 6 5 | 4 0 |
| D5..3 | Rc5 | S | Rb5 | 0 | Special5 | Md2 | D2..0 | Sz2 | 175 |
| Reglist32 | | | | | | | | | |
| Opt Constant31..0 | | | | | | | | | |
| Opt Constant63..32 | | | | | | | | | |

**Register List**

Each bit in the register list specifies the corresponding general purpose register to be loaded. Note that the zero register cannot be loaded.

**Special**

The special field specifies special purpose registers to be loaded. This list includes the loop counter, condition code group register and processor status register. Note that loading the status register is allowed only when the processor is in supervisor mode, otherwise a privilege violation will result.

Note that the PC, CCRG and SR are saved on the stack automatically for a supervisor mode processed exception. For an application exception the PC and CCRG are automatically saved on the stack.

|  |  |  |
| --- | --- | --- |
| Bit | Reg | Reg Description |
| 0 | PC | Program Counter |
| 1 | CCRG | Condition Codes Register Group |
| 2 | SR | Status Register |
| 3 | AC | Application Control Register |
| 4 | LC | Loop Counter |

Notes:

### PEA <ea>

**Description:**

Push effective address. The calculated address is pushed onto the stack. Note that pre and post updated instructions calculate an effective address but do not update the register.

**Supported Operand Sizes:** .t, .o

**Supported Address Modes:**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| Rn | D(Rn) | D(Rn)+ | +D(Rn) | D(Rn, Rn) | RFU | RFU | Imm |
|  | \* | \* | \* | \* |  |  |  |

**Instruction Formats: D(Rn), D(Rn)+, +D(Rn)**

If displacement bits 2 to 9 equal 80h then the displacement is 32 bits in the following instruction word. If displacement bits 2 to 9 equal 81h then the displacement is 64-bits in the following two instruction words. The value 82h in displacement bits 2 to 9 is reserved and should not be used.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 31 24 | 23 | 22 18 | 17 | 16 12 | 11 9 | 8 7 | 6 5 | 4 0 |
| Disp9..2 | ~ | Rb5 | 0 | ~5 | Md3 | D1..0 | Sz2 | 275 |
| Opt Constant31..0 | | | | | | | | |
| Opt Constant63..32 | | | | | | | | |

**Instruction Format: D(Rn, Rn)**

IF the displacement D bits equal 16 then the next word is used as the displacement. If the displacement D bits equal 17 then the next two words are used as the displacement. A D displacement bits equal to 18 are reserved.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 31 29 | 28 24 | 23 | 22 18 | 17 | 16 12 | 11 9 | 8 7 | 6 5 | 4 0 |
| D4..2 | Rc5 | ~ | Rb5 | 0 | ~5 | Md3 | D1..0 | Sz2 | 275 |
| Opt Constant31..0 | | | | | | | | | |
| Opt Constant63..32 | | | | | | | | | |

Notes:

### PUSH Rn

**Description:**

Push register Ra onto stack. This is an alternate mnemonic for the [STORE](#_STORE_Rn,<dea>) instruction where the destination is a pre-decremented stack pointer value.

**Supported Operand Sizes:** .t, .o

**Supported Address Modes:**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| Rn | D(Rn) | D(Rn)+ | +D(Rn) | D(Rn, Rn) | RFU | RFU | Imm |
|  |  |  | \* |  |  |  |  |

**Instruction Formats: +D(Rn)**

If displacement bits 2 to 9 equal 80h then the displacement is 32 bits in the following instruction word. If displacement bits 2 to 9 equal 81h then the displacement is 64-bits in the following two instruction words. The value 82h in displacement bits 2 to 9 is reserved and should not be used.

For a PUSH the displacement bits should be equal to -4 or -8.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 31 24 | 23 | 22 18 | 17 | 16 12 | 11 9 | 8 7 | 6 5 | 4 0 |
| Disp9..2 | ~ | Rb5 | 0 | Rn5 | 33 | D1..0 | Sz2 | 185 |

Notes:

### STORE Rn,<dea>

**Description:**

Store register Ra to destination.

**Supported Operand Sizes:** .b, .w, .t, .o

**Supported Address Modes:**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| Rn | D(Rn) | D(Rn)+ | +D(Rn) | D(Rn, Rn) | RFU | RFU | Imm |
|  | \* | \* | \* | \* |  |  |  |

**Instruction Formats: D(Rn), D(Rn)+, +D(Rn)**

If displacement bits 2 to 9 equal 80h then the displacement is 32 bits in the following instruction word. If displacement bits 2 to 9 equal 81h then the displacement is 64-bits in the following two instruction words. The value 82h in displacement bits 2 to 9 is reserved and should not be used.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 31 24 | 23 | 22 18 | 17 | 16 12 | 11 9 | 8 7 | 6 5 | 4 0 |
| Disp9..2 | C | Rb5 | 0 | Ra5 | Md3 | D1..0 | Sz2 | 185 |
| Opt Constant31..0 | | | | | | | | |
| Opt Constant63..32 | | | | | | | | |

**Instruction Format: D(Rn, Rn)**

IF the displacement D bits equal 16 then the next word is used as the displacement. If the displacement D bits equal 17 then the next two words are used as the displacement. A D displacement bits equal to 18 are reserved.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 31 29 | 28 24 | 23 | 22 18 | 17 | 16 12 | 11 9 | 8 7 | 6 5 | 4 0 |
| D4..2 | Rc5 | C | Rb5 | 0 | Ra5 | Md3 | D1..0 | Sz2 | 185 |
| Opt Constant31..0 | | | | | | | | | |
| Opt Constant63..32 | | | | | | | | | |

Notes:

### STORE #imm,<dea>

**Description:**

Store immediate constant to memory.

**Supported Operand Sizes:** .b, .w, .t, .o

**Supported Address Modes:**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| Rn | D(Rn) | D(Rn)+ | +D(Rn) | D(Rn, Rn) | RFU | RFU | Imm |
|  | \* | \* | \* | \* |  |  |  |

**Instruction Formats: D(Rn), D(Rn)+, +D(Rn)**

If displacement bits 2 to 9 equal 80h then the displacement is 32 bits in the following instruction word. If displacement bits 2 to 9 equal 81h then the displacement is 64-bits in the following two instruction words. The value 82h in displacement bits 2 to 9 is reserved and should not be used.

If the imm5 bits equal 16 then the next instruction word is a 32-bit immediate value. IF the imm5 bits equal 17 then the next two instruction words are a 64-bit value. The imm5 value of 18 is reserved and should not be used.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 31 24 | 23 | 22 18 | 17 | 16 12 | 11 9 | 8 7 | 6 5 | 4 0 |
| Disp9..2 | C | Rb5 | 1 | Imm5 | Md3 | D1..0 | Sz2 | 185 |
| Opt Imm31..0 | | | | | | | | |
| Opt Imm63..32 | | | | | | | | |
| Opt Constant31..0 | | | | | | | | |
| Opt Constant63..32 | | | | | | | | |

**Instruction Format: D(Rn, Rn)**

IF the displacement D bits equal 16 then the next word is used as the displacement. If the displacement D bits equal 17 then the next two words are used as the displacement. A D displacement bits equal to 18 are reserved.

If the imm5 bits equal 16 then the next instruction word is a 32-bit immediate value. IF the imm5 bits equal 17 then the next two instruction words are a 64-bit value. The imm5 value of 18 is reserved and should not be used.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 31 29 | 28 24 | 23 | 22 18 | 17 | 16 12 | 11 9 | 8 7 | 6 5 | 4 0 |
| D4..2 | Rc5 | C | Rb5 | 1 | Imm5 | Md3 | D1..0 | Sz2 | 185 |
| Opt Imm31..0 | | | | | | | | | |
| Opt Imm63..32 | | | | | | | | | |
| Opt Constant31..0 | | | | | | | | | |
| Opt Constant63..32 | | | | | | | | | |

Notes:

### STORE Reglist,<dea>

**Description:**

Store multiple registers to memory.

If bit ‘S’ of the instruction is set then the store addresses for registers not specified in the register list but does not store them. This allows values to memory to be skipped over. This could be used for storing thread state to a thread control block, omitting some of the registers from the store. Registers not specified are ‘skipped’ over, otherwise if ‘S’ is not set then registers specified are stored to consecutive memory locations.

**Supported Operand Sizes:** .b, .w, .t, .o

**Supported Address Modes:**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| Rn | D(Rn) | D(Rn)+ | +D(Rn) | D(Rn, Rn) | RFU | RFU | Imm |
|  | \* | \* | \* | \* |  |  |  |

**Instruction Formats: D(Rn), D(Rn)+, +D(Rn)**

If displacement bits 2 to 9 equal 80h then the displacement is 32 bits in the following instruction word. If displacement bits 2 to 9 equal 81h then the displacement is 64-bits in the following two instruction words. The value 82h in displacement bits 2 to 9 is reserved and should not be used.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 31 24 | 23 | 22 18 | 17 | 16 12 | 11 9 | 8 7 | 6 5 | 4 0 |
| Disp9..2 | S | Rb5 | 0 | Special5 | Md3 | D1..0 | Sz2 | 195 |
| Reglist32 | | | | | | | | |
| Opt Constant31..0 | | | | | | | | |
| Opt Constant63..32 | | | | | | | | |

**Instruction Format: D(Rn, Rn)**

IF the displacement D bits equal 16 then the next word is used as the displacement. If the displacement D bits equal 17 then the next two words are used as the displacement. A D displacement bits equal to 18 are reserved.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 31 29 | 28 24 | 23 | 22 18 | 17 | 16 12 | 11 9 | 8 7 | 6 5 | 4 0 |
| D4..2 | Rc5 | S | Rb5 | 0 | Special5 | Md3 | D1..0 | Sz2 | 195 |
| Reglist32 | | | | | | | | | |
| Opt Constant31..0 | | | | | | | | | |
| Opt Constant63..32 | | | | | | | | | |

**Special**

The special field specifies special purpose registers to be loaded. This list includes the loop counter, condition code group register and processor status register. Note that loading the status register is allowed only when the processor is in supervisor mode, otherwise a privilege violation will result.

Note that the PC, CCRG and SR are saved on the stack automatically for a supervisor mode processed exception. For an application exception the PC and CCRG are automatically saved on the stack.

|  |  |  |
| --- | --- | --- |
| Bit | Reg | Reg Description |
| 0 | PC | Program Counter |
| 1 | CCRG | Condition Codes Register Group |
| 2 | SR | Status Register |
| 3 | AC | Application Control Register |
| 4 | LC | Loop Counter |

Notes: