### 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 |  |
|  | CCR | Condition Codes Register |  |
|  | SR | Status Register |  |

### Condition Codes Register

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 eight flags which reflect the status of operations. Most instructions can select a condition register to update.

**Condition Register Group Register**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 63 56 | 55 48 | 47 40 | 39 32 | 31 24 | 23 16 | 15 8 | 7 0 |
| Cr7 | Cr6 | Cr5 | Cr4 | Cr3 | Cr2 | Cr1 | Cr0 |

**Condition Register Format:**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| ~ | ord | eq | eq | nf | zf | vf | cf |

### Status Register

The processor status register holds bits controlling the overall operation of the processor.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 31 24 |  |  |  |  |  |  |  |  | 15 |  | 13 |  | 11 | 10 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| CPL |  |  |  |  |  |  |  |  | T |  | S |  |  | IPL |  |  |  |  | D | D |  |  |

### Decimal Mode

Setting the ‘d’ flag bit 3 in the status 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 2, may also be applied to floating-point which will use decimal floating-point operations instead of binary.

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 |
|  |  |

### Major Opcode

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| 0x | 0  BRK | 1  {R1} | 2  {CSR} | 3  CMP | 4  ADD | 5  SUB | 6  MUL | 7  DIV |
|  | 8  AND | 9  OR | 10  XOR | 11 | 12 | 13 | 14  SHIFT | 15  {Misc} |
| 1x | 16  LOAD | 17  LOADM | 18  STORE | 19  STOREM | 20  FMA | 21  FMS | 22  FNMA | 23  FNMS |
|  | 24  CALL abs | 25 | 26 | 27 | 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.

**Address Modes**

|  |  |  |  |
| --- | --- | --- | --- |
| **Address Mode** | **Format** | **M3** | **Reg5** |
| Register direct | Rn | 0 | Rn |
| Register indirect with displacement | Disp(Rn) | 1 | Rn |
| Post Update Register indirect with displacement | Disp(Rn)+ | 2 | Rn |
| Pre Update Register indirect with displacement | +Disp(Rn) | 3 | Rn |
| Indexed1 | (Rn,Rn) | 4 | Rn |
| Indexed with 32-bit displacement1 | Disp(Rn,Rn) | 5 | Rn |
| Indexed with 64-bit displacement1 | Disp(Rn,Rn) | 6 | Rn |
| Immediate Nine | Imm9 | 7 | 0nnnn |
| Immediate Thirty-Two | Imm32 | 7 | 10000 |
| Immediate Sixty-Four | Imm64 | 7 | 10001 |

1For indexed modes, R0 refers to the program counter

Three-bit displacement is in terms of operation size units.

**Operation Size**

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

Operand Swap

|  |  |
| --- | --- |
| **Operand Order** | **S** |
| Mem is 2nd source operand | 0 |
| Mem is 1st source operand | 1 |

Status Register

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 31 24 | 23 21 | 20 16 | 15 | 14 | 13 | 12 | 11 | 10 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| CPL8 | ~3 | Rr5 | tf | ~ | sf | rf | ~ | Im3 | ~ | ~ | df | xf | ~ | ~ | ~ | ~ |

df: decimal arithmetic flag

tf: trace mode

### 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.

**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 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: 6**

**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: 8**

### 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.

**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 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: 6**

**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: 8**

### 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.

**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: 6**

**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: 8**

**Clock Cycles:**

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

**Exceptions:** none

**Notes:**

### 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.

**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 24 | 23 | 22 18 | 17 | 16 12 | 11 7 | 6 5 | 4 0 |
| Ct3 | 02 | ~3 | 0 | 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 | 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.

**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 |

**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:**

**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 and as floating-point values.

**Condition Register Format:**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| ~ | ord | lt | eq | nf | zf | vf | cf |

**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:**

Rt = Ra ? Rb or Rt = Ra ? Imm

**Clock Cycles:**

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

**Exceptions:** none

**Notes:**

**Instruction Formats:**

**CMP 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 | ~5 | Sz2 | 35 |

**CMP 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 | ~5 | Sz2 | 35 |

**CMP 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 | ~5 | Sz2 | 35 |
| Immediate32 | | | | | | | | |

**CMP 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 | ~5 | 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 |  |

|  |  |  |
| --- | --- | --- |
| **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 | Sz2 | 26 |

### DIV - 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.

**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:**

**DIV 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 |

**DIV 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 |

**DIV 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 | | | | | | | | |

**DIV 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.

**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.

**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:**

### 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.

**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 24 | 23 | 22 18 | 17 | 16 12 | 11 7 | 6 5 | 4 0 |
| Ct3 | 12 | ~3 | 0 | Rb5 | 01 | Ra5 | Rt5 | Sz2 | 145 |

**LSL Rt,Ra,Imm6**

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 31 29 | 28 27 | 2625 | 24 | 23 | 22 18 | 17 | 16 12 | 11 7 | 6 5 | 4 0 |
| Ct3 | 12 | ~2 | 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.

**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 | 12 | ~3 | 1 | Rb5 | 01 | Ra5 | Rt5 | Sz2 | 145 |

**ASL Rt,Ra,Imm6**

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

**Clock Cycles:**

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

**Exceptions:** none

**Notes:**

### MOVE <dea>,<sea>

**Description:**

Move data from source to destination.

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

**Supported Address Modes:**

Source:

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

Destination:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Rn | D(Rn) | D(Rn)+ | -D(Rn) | (Rn, Rn) | D32(Rn,Rn) | D64(Rn,Rn) | Imm |
| \* | \*1 | \*1 | \*1 | \* | \* | \* |  |

1Register indirect with three-bit displacement is not supported for destination operands.

**Instruction Format:**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 31 27 | 26 | 25 21 | 20 18 | 17 13 | 12 10 | 9 8 | 7 6 | 5 0 |
| D5 | C | Rb5 | Ms3 | Ra5 | Md3 | Ds | Sz2 | 166 |
| Opt Src Constant31..0 | | | | | | | | |
| Opt Src Constant63..32 | | | | | | | | |
| Opt Dst Constant31..0 | | | | | | | | |
| Opt Dst Constant63..32 | | | | | | | | |

Notes:

### MOVEM – Move Multiple Registers

LOAD reglist, <sea>

STORE reglist, <dea>

**Description:**

Move data from source to destination.

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

**Supported Address Modes:**

Source:

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

Destination:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Rn | D(Rn) | D(Rn)+ | -D(Rn) | (Rn, Rn) | D32(Rn,Rn) | D64(Rn,Rn) | Imm |
| \* | \*1 | \*1 | \*1 | \* | \* | \* |  |

1Register indirect with three-bit displacement is not supported for destination operands.

**Instruction Format:**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 31 27 | 26 | 25 21 | 20 18 | 17 13 | 12 10 | 9 8 | 7 6 | 5 0 |
| D5 | C | Rb5 | Ms3 | Cr List8 | | ~ | Sz2 | 166 |
| Reglist31..0 | | | | | | | | |
| Opt Src/Dst Constant31..0 | | | | | | | | |
| Opt Src/Dst Constant63..32 | | | | | | | | |

Notes:

### MUL - 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.

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

**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:**

**MUL 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 |

**MUL 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 |

**MUL 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 | | | | | | | | |

**MUL 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.

**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.

**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:**

### ROL – Rotate Left

**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.

**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 24 | 23 | 22 18 | 17 | 16 12 | 11 7 | 6 5 | 4 0 |
| Ct3 | 22 | ~3 | 0 | Rb5 | 01 | Ra5 | Rt5 | Sz2 | 145 |

**ROL Rt,Ra,Imm6**

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 31 29 | 28 27 | 2625 | 24 | 23 | 22 18 | 17 | 16 12 | 11 7 | 6 5 | 4 0 |
| Ct3 | 22 | ~2 | 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.

**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 24 | 23 | 22 18 | 17 | 16 12 | 11 7 | 6 5 | 4 0 |
| Ct3 | 22 | ~3 | 1 | Rb5 | 01 | Ra5 | Rt5 | Sz2 | 145 |

**ROR Rt,Ra,Imm6**

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

**Clock Cycles:**

**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.

**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**

### 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.

**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 |

### RTI – Return From Interrupt

**Instruction Formats:**

**RTS #Arg,#Rpt**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 31 29 | 28 24 | 23 | 22 18 | 17 | 16 12 | 11 7 | 6 5 | 4 0 |
| Ct3 | 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 then the program counter from the stack. Add Rpt words to the program counter, and Arg words 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 |
| Ct3 | 05 | 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.

Arg11..2 is the number of words of subroutine arguments to remove from the stack. Up to 1024 words 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. Note it is possible to execute traps beyond number 31 by placing the trap number (0 to 223) in register Ra.

**Instruction Format:**

**TRAP Rt, Ra, #Data**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 31 29 | 28 18 | 17 | 16 12 | 11 7 | 6 5 | 4 0 |
| Ct3 | Data10..0 | 01 | Ra5 | Rt5 | 02 | 05 |

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

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 31 29 | 28 18 | 17 | 16 12 | 11 7 | 6 5 | 4 0 |
| Ct3 | Data10..0 | 11 | Num5 | Rt5 | 02 | 05 |

**Operation:**

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

### 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.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 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.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 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.

**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 | Special5 | Md3 | D1..0 | Sz2 | 185 |
| 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 | C | Rb5 | 0 | Special5 | Md3 | D1..0 | Sz2 | 185 |
| Reglist32 | | | | | | | | | |
| Opt Constant31..0 | | | | | | | | | |
| Opt Constant63..32 | | | | | | | | | |

Notes: