FT64v8c

## Rational

Why yet another architecture? Depending on a single architecture increases the odds that if something goes amiss with it, everyone is affected. Nature does not have a single architecture.

## The Beauty of ISA’s

Some of the best ISA’s aren’t very attractive, they have all kinds of features that the novice user asks: why did they do that? The ISA’s are not very attractive because they are functional and serve a purpose for which they were engineered.

## Separation of Register Sets

One of the things the 68k architecture does right is separate data and address calculations. Address calculations don’t require the same number and variability of operations as data calculations. Many modern processors have explicit address generation units separate from the data calculation ALU. FT64 takes this one step further and separates code and data addresses. Many modern processors contain branch calculation units, or flow control units. It appears to the author that the register set depends on the functional unit although it doesn’t have to. Floating point units often come with their own register set.

The PowerPC has several features that the author likes. One of the things the PowerPC architecture gets right is the provision for multiple condition code registers. A second feature the of PowerPC is indexed addressing.

Separating out sets of registers specific to functional units may make it possible to get away with fewer register ports, in particular register write ports, which tend to be expensive.

## Addressing Modes

The problem with indexed addressing is that when it’s needed it’s needed. Sure, it can be emulated using several simpler instructions and additional registers, but why not just include it directly in the ISA? There are code density benefits to supporting indexed addressing, if not clock cycle benefits. The base addressing mode supported by FT64 is indexed with displacement addressing mode. Other modes can be emulated as a subset of that mode.

## Compressed Instruction Sets

Compressed instruction sets are in the author’s view a kludge to increase code density. What’s really needed are variable instruction lengths. Part of the appeal of a compressed instruction set is that it makes the calculation of the instruction length simple. When active, compressed instructions are usually a fixed length. FT64v8 will have variable instruction lengths and the length of the instruction will be determined by the first byte of the instruction.

# Programming Model

# Registers

One thing that seems to be clear in modern computers is that memory is much slower than the cpu. This means that caching of values becomes extremely important. Machine registers are the level zero cache for values from memory. Having adequate types and numbers of registers is important.

## Code Address Registers (CA0 to CA7)

The processor contains eight code address registers (CA0-CA7). Several of the registers are reserved for predefined purposes. A code address register is used in the formation and storage of code addresses. Code address registers may hold branch targets.

|  |  |  |
| --- | --- | --- |
| Reg |  | Usage |
| RA0 |  | Subroutine return address |
| RA1 |  |  |
| LC |  |  |
| LC |  |  |
|  |  |  |
| CL | Catch Link Register | Used by the compiler to link to try/catch handlers. |
| XP | Exceptioned PC | This register is automatically set during a hardware interrupt or exception |
| PC | Program Counter | Relative address formation. |

Code address registers may be used to point to a block of code from which the JSR instruction can index into with its 24-bit offset. For instance, a register may contain a pointer to a class method jump list; the JSR instruction can then index into this list in order to invoke a method.

The presence of multiple code address registers allows multi-level return addresses to be used for performance. Leaf routines may use CA0 as the return address. Next to leaf routines may use CA1, etc. So that memory operations are avoided when implementing subroutine call and return.

The program counter register is read-only. The program counter cannot be modified by moving a value to this register.

## Program Counter (CA7)

|  |  |
| --- | --- |
| 63 40 | 39 0 |
| Program Bank | Program Counter |

The program counter is special in that it is always incrementing by the size of the instructions fetched as a program runs. Program code is byte aligned. To improve performance only the low order 40 bits of the program counter increment. The entire program counter may be loaded with a jump instruction. If the upper six bits of the program counter/ bank are all ones, then segmentation with the code segment is ignored.

## Address Registers (A0 to A15)

Data address registers are used to locate data as opposed to instructions.

|  |  |  |
| --- | --- | --- |
| Reg | 63 0 |  |
| A0 | always zero |  |
| A1 | return value |  |
| A2 |  |  |
| A3 |  |  |
| A4 |  |  |
| A5 |  |  |
| A6 |  |  |
| A7 |  |  |
| A8 |  |  |
| A9 |  |  |
| A10 |  |  |
| A11 |  |  |
| A12 |  |  |
| A13 | Frame Pointer |  |
| A14 | Stack Pointer |  |
| A15 | Program Counter |  |

## Condition Code (CC0 to CC7)

There are eight condition code registers. Each condition code register contains eight flags.

|  |  |
| --- | --- |
| Reg |  |
| CC0 |  |
| CC1 |  |
| CC2 |  |
| CC3 |  |
| CC4 |  |
| CC5 |  |
| CC6 |  |
| CC7 |  |

|  |  |  |
| --- | --- | --- |
| Bit | Flag | Meaning |
| 0 | Z | zero |
| 1 | N | negative |
| 2 | V | overflow |
| 3 | C | carry |
| 4 | O | odd |
| 5 | P | parity |
| 6 |  | reserved |
| 7 | U | unordered |

## Data Registers (D0 to D31)

|  |  |  |
| --- | --- | --- |
| Reg # |  |  |
| 0 | always zero |  |
| 1 | return value |  |
| 2 |  |  |
| 3 |  |  |
| 4 |  |  |
| 5 |  |  |
| 6 |  |  |
| 7 |  |  |
| 8 |  |  |
| 9 |  |  |
| 10 |  |  |
| 11 |  |  |
| 12 |  |  |
| 13 |  |  |
| 14 |  |  |
| 15 |  |  |
| 16 |  |  |
| 17 |  |  |
| 18 |  |  |
| 19 |  |  |
| 20 |  |  |
| 21 |  |  |
| 22 |  |  |
| 23 |  |  |
| 24 |  |  |
| 25 |  |  |
| 26 |  |  |
| 27 |  |  |
| 28 |  |  |
| 29 |  |  |
| 30 |  |  |
| 31 |  |  |

## Segment Registers (ZS, DS, ES, FS, GS, HS, SS, CS)

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  | ZS |  |
|  | DS | data segment |
|  | ES |  |
|  | FS |  |
|  | GS |  |
|  | HS |  |
|  | SS | stack segment |
|  | CS | code segment |
|  | TR | thread register |
|  | LDT | local descriptor table |
|  | GDT | global descriptor table |
|  | IDT | interrupt descriptor table |
|  | ID | interrupt temporary |

# Instruction Set Description

## ADD - Add

**Description**:

**Instruction Format**:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 23 | 22 18 | 17 13 | 12 8 | 7 0 |
| ~ | Db5 | Da5 | Dt5 | 24h8 |

**Instruction Format**:

|  |  |  |  |
| --- | --- | --- | --- |
| 31 18 | 17 13 | 12 8 | 7 0 |
| Immediate14 | Da5 | Dt5 | 04h8 |

**Instruction Format**:

|  |  |  |  |
| --- | --- | --- | --- |
| 47 18 | 17 13 | 12 8 | 7 0 |
| Immediate30 | Da5 | Dt5 | 14h8 |

## ADDA – Add to Address

**Description**:

An immediate value is multiplied by a scaling factor then added to the address register.

**Instruction Format**:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 23 | 22 18 | 17 13 | 12 8 | 7 0 |
| ~ | Db5 | Aa5 | At5 |  |

**Instruction Format**:

|  |  |  |  |
| --- | --- | --- | --- |
| 31 22 | 21 12 | 11 8 | 7 0 |
| Immediate10 | Sc10 | At4 |  |

|  |  |  |  |
| --- | --- | --- | --- |
| 47 28 | 27 12 | 11 8 | 7 0 |
| Immediate20 | Sc16 | At4 |  |

## BEQ – Branch if Equal

**Description**:

Branch if the condition code register indicates equality.

**Target Address Calculation**:

The lower 12 bits of the target address come from the offset field in the instruction. The middle 20 bits of the target address come from the sum of sign extended displacement field of the instruction shifted left by 12 bits and the code address register bits 12 to 31. The upper 32 bits of the target address come from the upper 32 bits of the code address register.

The program counter register may be used in the calculation.

**Instruction Format (DPO)**:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 31 29 | 28 24 | 23 12 | 11 | 10 8 | 7 0 |
| CA3 | Disp5 | Offset12 | ~ | CC3 | 36h8 |

## BNE – Branch if Not Equal

**Description**:

Branch if the condition code register indicates inequality.

**Target Address Calculation**:

The lower 12 bits of the target address come from the offset field in the instruction. The middle 20 bits of the target address come from the sum of sign extended displacement field of the instruction shifted left by 12 bits and the code address register bits 12 to 31. The upper 32 bits of the target address come from the upper 32 bits of the code address register.

**Instruction Format (DPO)**:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 31 29 | 28 24 | 23 12 | 11 | 10 8 | 7 0 |
| CA3 | Disp5 | Offset12 | ~ | CC3 | 37h8 |

## BRA – Branch Always

**Description**:

Unconditionally branch.

**Target Address Calculation**:

The lower 12 bits of the target address come from the offset field in the instruction. The middle 20 bits of the target address come from the sum of sign extended displacement field of the instruction shifted left by 12 bits and the code address register bits 12 to 31. The upper 32 bits of the target address come from the upper 32 bits of the code address register.

**Instruction Format (DPO)**:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 31 29 | 28 24 | 23 12 | 11 | 10 8 | 7 0 |
| CA3 | Disp5 | Offset12 | ~ | ~3 | 35h8 |

## CMP - Compare

**Description**:

**Instruction Format**:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 2322 | 21 17 | 16 12 | 11 | 10 8 | 7 0 |
| ~ | Db4 | Da5 | ~ | CC3 |  |

**Instruction Format**:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 31 17 | 16 12 | 11 | 10 8 | 7 0 |
| Immediate15 | Da5 | ~ | CC3 |  |

**Instruction Format**:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 47 17 | 16 12 | 11 | 10 8 | 7 0 |
| Immediate31 | Da5 | ~ | CC3 |  |

## CMPA – Compare Addresses

**Description**:

Compare two address values.

**Instruction Format**:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 23 20 | 19 16 | 15 12 | 11 | 10 8 | 7 0 |
| ~ | Ab4 | Aa4 | ~ | CC3 |  |

LW – Load word

Description:

Instruction Format:

|  |  |  |  |
| --- | --- | --- | --- |
| 15 13 | 12 8 | 7 0 |  |
| Disp3 | Dt5 |  | LDW Dt,d[SP] |

|  |  |  |  |
| --- | --- | --- | --- |
| 15 13 | 12 8 | 7 0 |  |
| Disp3 | Dt5 |  | LDW Dt,d[FP] |

|  |  |  |  |
| --- | --- | --- | --- |
| 15 12 | 11 8 | 7 0 |  |
| Disp4 | At4 |  | LA At,d[SP] |

|  |  |  |  |
| --- | --- | --- | --- |
| 15 12 | 11 8 | 7 0 |  |
| Disp4 | At4 |  | LA At,d[FP] |

|  |  |  |  |
| --- | --- | --- | --- |
| 23 13 | 12 8 | 7 0 |  |
| Disp11 | Dt5 |  | LDW Dt,d[SP] |

|  |  |  |  |
| --- | --- | --- | --- |
| 23 13 | 12 8 | 7 0 |  |
| Disp11 | Dt5 |  | LDW Dt,d[FP] |

|  |  |  |  |
| --- | --- | --- | --- |
| 23 12 | 11 8 | 7 0 |  |
| Disp12 | At4 |  | LA At,d[SP] |

|  |  |  |  |
| --- | --- | --- | --- |
| 23 12 | 11 8 | 7 0 |  |
| Disp12 | At4 |  | LA At,d[FP] |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 31 20 | 19 17 | 16 13 | 12 8 | 7 0 |  |
| Disp12 | Seg3 | Aa4 | Dt5 |  | LDW Dt,d[An] |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 31 20 | 19 17 | 16 13 | 12 | 11 8 | 7 0 |  |
| Disp12 | Seg3 | Aa4 | ~ | At4 |  | LW At,d[An] |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 31 26 | 25 | 24 20 | 19 17 | 16 13 | 12 8 | 7 0 |  |
| Offset6 | AD | Xn5 | Seg3 | Aa4 | Dt5 |  | LW Dt,o[An+Xn] |

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 31 26 | 25 | 24 20 | 19 17 | 16 13 | 12 | 11 8 | 7 0 |  |
| Offset6 | AD | Xn5 | Seg3 | Aa4 | ~ | At4 |  | LW At,o[An+Xn] |

## LD – Load Double Word (64 bits)

**Description**:

Loads a double-word (64-bit) value into a data register using indexed addressing. The primary instruction format is one of indexed addressing plus displacement. Four registers are summed in order to calculate the linear address. An address register, an index register, an offset from the instruction and a segment base register.

There are few additional forms of this instruction that make assumptions about the registers in use in order to shorten the instruction.

**Instruction Format**:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 47 26 | 25 | 24 20 | 19 17 | 16 13 | 12 8 | 7 0 |
| Offset22 | AD | Xn5 | Seg3 | Aa4 | Dt5 |  |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 47 26 | 25 | 24 20 | 19 17 | 16 13 | 12 | 11 8 | 7 0 |
| Offset22 | AD | Xn5 | Seg3 | Aa4 | ~ | At4 |  |

**Instruction Format**:

These two instruction formats are the same as the previous two except with only a six bit displacement field rather than twenty-two bits.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 31 26 | 25 | 24 20 | 19 17 | 16 13 | 12 8 | 7 0 |
| Offset6 | AD | Xn5 | Seg3 | Aa4 | Dt5 |  |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 31 26 | 25 | 24 20 | 19 17 | 16 13 | 12 | 11 8 | 7 0 |
| Offset22 | AD | Xn5 | Seg3 | Aa4 | ~ | At4 |  |

**Instruction Format**:

This instruction formats assumes the stack pointer is used in the address calculation and assumes the stack segment is also used.

|  |  |  |  |
| --- | --- | --- | --- |
| 23 13 | 12 8 | 7 0 |  |
| Disp11 | Dt5 |  | LD Dt,d[SP] |

**Instruction Format**:

This instruction formats assumes the frame pointer is used in the address calculation and assumes the stack segment is also used.

|  |  |  |  |
| --- | --- | --- | --- |
| 23 13 | 12 8 | 7 0 |  |
| Disp11 | Dt5 |  | LD Dt,d[FP] |

**Instruction Format**:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 31 20 | 19 17 | 16 13 | 12 8 | 7 0 |  |
| Disp12 | Seg3 | Aa4 | Dt5 |  | LD Dt,d[An] |

## LDIS – Load Immediate into Selector

Description:

Instruction Format:

|  |  |  |  |
| --- | --- | --- | --- |
| 31 12 | 11 | 10 8 | 7 0 |
| Immediate20 | ~ | Sg3 |  |

## LUI – Load Upper Immediate

**Description**:

**Instruction Format**:

Loads bits 29 to 63 with a sign extended immediate constant and zeros out the lower 29 bits of the target register. A subsequent ADD or OR instruction may be used to build a 48-bit constant in a register.

|  |  |  |
| --- | --- | --- |
| 31 13 | 12 8 | 7 0 |
| Immediate19 | Da5 |  |

**Instruction Format**:

Loads the upper 35 bits of a register with an immediate constant and zeros out the lower 29 bits of the target register. A subsequent ADD or OR instruction may be used to build a 64-bit constant in a register.

|  |  |  |
| --- | --- | --- |
| 47 13 | 12 8 | 7 0 |
| Immediate35 | Dt5 | 04h8 |

PTRDIF

Description:

This instruction converts a difference between two pointers into an index. Compute the difference between two pointers and shift the result right.

Instruction Format:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 23 | 22 20 | 19 16 | 15 12 | 11 7 | 6 0 |
|  | Sc3 | Ab4 | Aa4 | Dt5 |  |

|  |  |
| --- | --- |
| SC3 | Pointer Difference |
| 0 | byte pointer |
| 1 | char pointer |
| 2 | half pointer |
| 3 | word pointer |
| 4 | dword pointer |
| 5 | quad word pointer |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | x0 | x1 | x2 | x3 | x4 | x5 | x6 | x7 | x8 | x9 | xA | xB | xC | xD | xE | xF |
| 0x32 |  |  |  | CMP #14 | ADD #14 |  |  |  | AND #14 | OR #14 | XOR #14 |  |  |  | XNOR #14 |  |
| 1x48 |  |  |  | CMP #30 | ADD #30 |  |  |  | AND #30 | OR #30 | XOR #30 |  |  |  |  |  |
| 2x24 |  |  |  | CMP | ADD |  |  |  | AND | OR | XOR |  |  |  | XNOR |  |
| 3x | JMP | CALL | RET | BRK |  | BRA | BEQ | BNE | BLT | BGE | BLE | BGT | BLTU | BGEU | BLEU | BGTU |
| 4x |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 5x |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 6x |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 7x |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 8x | LB | LBU | LH | LHU | LW | LWU | LD48 Dn | LDU | LQ | LQU | LO |  | LD48 An |  |  |  |
| 9x |  |  |  |  |  |  | LD32 Dn |  |  |  |  |  | LD32 An |  |  |  |
| Ax |  |  |  |  |  |  | LD24 dSP |  |  |  |  |  | LD24 dSP |  |  |  |
| Bx |  |  |  |  |  |  | LD24 dFP |  |  |  |  |  | LD24 dFP |  |  |  |
| Cx | SB | SH | SW | SD | SQ | SO |  |  |  |  |  |  |  |  |  |  |
| Dx |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Ex |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Fx |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |