robfinch@finitron.ca

An overview of the FT833 CPU Core. Includes documentation on core register set, core instructions, parameters and configuration.

FT833 CPU Core

Table of Contents

[Overview 4](#_Toc484529347)

[Features 4](#_Toc484529348)

[Programming Model 5](#_Toc484529349)

[New Registers 7](#_Toc484529350)

[Status Register Extension 7](#_Toc484529351)

[Status Register 7](#_Toc484529352)

[Operating Modes 8](#_Toc484529353)

[Instruction Cache 9](#_Toc484529354)

[Assembler Notations 10](#_Toc484529355)

[New Addressing Modes 10](#_Toc484529356)

[Instruction Set Summary 11](#_Toc484529357)

[What’s Covered 11](#_Toc484529358)

[Timing 11](#_Toc484529359)

[AAX – Add Accumulator and X 11](#_Toc484529360)

[ASR – Arithmetic Shift Right 12](#_Toc484529361)

[BGT – Branch if Greater Than 13](#_Toc484529362)

[BLE – Branch if Less or Equal 14](#_Toc484529363)

[BMC – Bitmap Clear 15](#_Toc484529364)

[BMS – Bitmap Set 16](#_Toc484529365)

[BMT – Bitmap Test 17](#_Toc484529366)

[BYT –Byte Operation Prefix 18](#_Toc484529367)

[CACHE 19](#_Toc484529368)

[CLI – Clear Interrupt Mask 20](#_Toc484529369)

[CMC – Compliment Carry 21](#_Toc484529370)

[DEX4 – Decrement .X by Four 22](#_Toc484529371)

[DEY4 – Decrement .Y by Four 23](#_Toc484529372)

[FIL 24](#_Toc484529373)

[INF - Information 26](#_Toc484529374)

[INX4 – Increment .X by Four 28](#_Toc484529375)

[INY4 – Increment .Y by Four 29](#_Toc484529376)

[LDO – Load Offset Register 30](#_Toc484529377)

[MUL - Multiply 31](#_Toc484529378)

[PHO – Push Offset Register 32](#_Toc484529379)

[PLO – Pull Offset Register 33](#_Toc484529380)

[RTI - Return From Interrupt 34](#_Toc484529381)

[RTL – Long Return From Subroutine 35](#_Toc484529382)

[RTS – Return From Subroutine 36](#_Toc484529383)

[SEI – Set Interrupt Mask Level 37](#_Toc484529384)

[TAO – Transfer .A to Program Offset 38](#_Toc484529385)

[TOA – Transfer Program Offset to .A 39](#_Toc484529386)

[XBAW – Exchange B and A Words 41](#_Toc484529387)

[Core Parameters 42](#_Toc484529388)

[Configuration Defines 43](#_Toc484529389)

[I/O Ports 44](#_Toc484529390)

[Opcode Map 46](#_Toc484529391)

# Overview

The design of this core has been guided by discussions on the 6502.org forum. Features of the core include truly flat 32 bit addressing and 32 bit indirect addresses. The core is 65832 backwards compatible. New instructions have been added to support core functionality. Some of the instruction set has been designed around the notion that this core will be required for more heavy duty apps.

# Features

Some features include:

Expanded addressing capabilities (32 bit addressing modes)

Instruction caching

Program offset register

Single step mode

Combinational signed branches (branches that test both N and Z flags at the same time).

Long branching for regular branch instructions

Multiply instruction

Enhanced support for variable size data (size prefix codes)

Expanded interrupt capabilities

# Programming Model

The programming model is compatible with the W65C816S programming model, with the addition of an offset register. A number of new instructions and addressing modes have been added using the opcode reserved for that purpose (the WDM opcode).

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Register | FT833 | Size |  |  |
| PO | \* | 32 | program offset |  |
| PB |  | 8 | program bank |  |
| PC |  | 16 | program counter |  |
| Acc |  | 32 | accumulator |  |
| x |  | 32 | x index register |  |
| y |  | 32 | y index register |  |
| SP |  | 32 | stack pointer |  |
| DB |  | 8 | data bank |  |
| DPR |  | 16 | direct page register |  |
| SR |  | 8 | status register |  |
| SRX | \* | 8 | status register extension |  |

Register Settings on Reset

|  |  |  |
| --- | --- | --- |
|  |  | Note: |
| CS | Zero | * reset to zero – required since the CS is not part of the reset vector |
| PB | $00 | * reset to zero – required since the PB is not part of the reset vector |
| PC | $FFF0 | * this register value will be overwritten and automatically loaded from the reset vector in memory on a reset |
| DS | --- | * not set by reset |
| DB | --- | “ |
| DPR | --- | “ |
| A |  | “ |
| X |  | “ |
| Y |  | “ |
| SS | --- | “ |
| SP | $000001FF | * since the stack page is being set to page 1, the remainder of the stack pointer is set as well |
| SR | %xx0x01xx | * interrupts are masked, and decimal mode is cleared (note the m and x bits are set but not visible as part of the status register because the core starts in eight bit emulation mode). |
| SRX | %xxx0x000 | * the emulation mode is set to eight bit, both the 32 and 16 bit emulation flags are cleared, interpreter mode and single step mode are disabled. |
| TR | $00 | * the task register identifies which task is running. It is an internal register, set indirectly by the TSK instruction. |
|  |  |  |

On reset the contents of the task context register array is undefined.

## New Registers

There is a new program offset register. The program offset register is added to addresses as a program runs. The default value of the offset register is zero.

The addition of these registers is a result of discussions on 6502.org. Forum members expressed a desire to have a full 32 bit program bank and data bank registers allowing the base address of the program or data to be placed anywhere in memory. Rather than modify the existing program bank and data bank registers, a new program offset was added. This allows the core to be backwards compatible with the 65816/65832 design. If desired the program bank and data bank registers may be set to zero, and the 32 bit program offset used to place code / data in memory. Alternately the program offset register could be set to zero and the core used as a 65816/65832 compatible core. There are new instructions ([PHO](#_PHO_–_Push), [PLO](#_PLO_–_Pull), [LDO](#_LDO_–_Load), [TAO](#_TAO_–_Transfer), [TOA](#_TOA_–_Transfer)) to support use of the offset register in a manner similar to the program bank and data bank registers.

There is an extension to the status register called the SRX register, which contains the emulation mode setting bits. The 65816/65832 doubles up on the usage of the C and V flags in the status register in order to set the processor mode. This approach was likely used in order to avoid creating another program visible register in the processor. This is acceptable because there isn’t really a need to store the emulation mode bits. A new register has been added in this design in order to support additional core options.

## Status Register Extension

|  |  |  |
| --- | --- | --- |
| Bit |  | Usage |
| 0 | m816 | 16 bit emulation mode flag |
| 1 | m832 | 32 bit native mode operation flag |
| 2 |  |  |
| 3 | ssm | single step mode |
| 4 | iml | 000 = no masking  111 = all ints masked |
| 5 |
| 6 |
| 7 | OS | offset register changed flag |

## Status Register

The setting of the interrupt mask flag now reflects whether the interrupt mask level is zero or some other level. This bit will be zero if interrupts are at level zero, otherwise the bit is a one.

# Operating Modes

# Instruction Cache

For better performance, memory is often organized in a hierarchy that consists of caches isolating the access to main memory. Caches are faster than main memory, and higher level caches (closest to the cpu) are faster than lower leveled ones. In the FT833 cpu all instruction accesses are cached. While this doesn’t necessarily result in better instruction execution performance for the intended target of the FT833 (a PLD), it does reduce the amount of traffic on the bus. This means that systems sharing the bus can have better performance as bus availability is increased. For instance the [TSK](#_TSK) instruction takes four cycles to execute, but doesn’t use the bus. Hence the bus is available for at least four consecutive clock cycles while the TSK instruction executes.

The default instruction cache is organized as 256, 16 byte lines. An entire cache line is loaded with back-to-back memory read operations as fast as the memory system will allow. The leading byte of an instruction cache line fetch is signified with both VPA and VDA signals being active. This is similar to the first byte of an opcode fetch being signified in the same manner on the 65816.

Cache lines may be pre-loaded so that the performance of specific code is not impacted by line loads. The cache may also be invalidated on a line-by-line basis, or the entire cache can be invalidated. Cache control is via the ‘[CACHE](#_CACHE)’ command instruction. Note that invalidating or pre-loading a cache line that conflicts with the current instruction’s cache line causes the instruction’s cache line to be reloaded from memory (otherwise the core wouldn’t be able to execute instructions). Care must be taken to place code such that cache line conflicts do not occur if it is desired to preload the cache lines.

The core uses a 16 byte window into the instruction cache from which instruction data is read. All 16 bytes are available in parallel within a single clock cycle. This means that the instruction fetch time is always fixed at a single clock cycle regardless of the length of an instruction. IT also means that an instruction including any prefixes cannot be longer than 16 bytes. The window slides as the program counter value changes. This window will usually span two cache lines. On occasion it may be necessary to fetch two lines from memory in order for an instruction spanning cache lines to execute.

The instruction cache is physically indexed and tagged. The cache is driven by the address resulting from the sum of the code segment and program counter. This results in only a single image of instructions in the cache when different combinations of the program counter and code segment result in the same address.

# Assembler Notations

Since the core supports 32 bit indirect addressing a new notation is required for assembler code. Thirty-two bit indirect addresses are denoted with { } characters. For instance to access data pointed to with a 32 bit indirect address: LDA {$23},Y

The FT833 core also has operand size control prefixes. These prefixes are specified by appending a dot code onto the instruction they apply to. For instance to apply the BYT prefix to the LDA instruction use the notation “LDA.B”.

|  |  |  |
| --- | --- | --- |
| Instruction Suffix |  |  |
| .B | signed byte operand |  |
| .UB | unsigned byte operand |  |
| .H | signed half-word (16 bit) operand |  |
| .UH | unsigned half-word (16 bit) operand |  |

# New Addressing Modes

There are several new addressing modes for existing instructions. Extra-long addressing for both absolute and absolute indexed addresses is available. The extra-long addressing mode is formed by prefixing the regular absolute address modes opcode with the extended opcode indicator byte ($42). This gives access to a 32 bit offset for a number of instructions which were not supported by the absolute long address modes. Extra-long indirect addressing modes are additional addressing mode available in the same manner as extra-long addressing. The indirect address mode instructions are prefixed with the opcode extension byte ($42).

# Instruction Set Summary

## What’s Covered

Only the enhanced instruction set instructions are documented here. Documentation on the remaining instructions which are W65C816S compatible is well done in the W65C816 programming manual. One notable difference is the instruction timings. The clock cycle counts for this core are not guaranteed to match those of a genuine 65C816.

## Timing

Instruction timings may be dependent on memory access time for those instructions which access memory. The clock cycles counts are assuming that memory can be accessed in a single cycle. In many systems this is not the case and the memory system will insert wait states. Internal states are performed in a single clock cycle. Instruction fetch is single cycle to retrieve all bytes associated with the instruction assuming the instruction is located in the cache.

## AAX – Add Accumulator and X

Add .X register to accumulator. This instruction is useful in calculating double indexed values. For example (X+Y) addressing: TYA, AAX, TAX.

**Opcode Format (2 bytes)**

|  |  |
| --- | --- |
| 42 | 8A |

3 clock cycles

The C, N, V, and Z flags are updated by this instruction.

**Machine States:**

|  |  |
| --- | --- |
| IFETCH | Fetch the instruction |
| DECODE | Decode the page 2 prefix |
| DECODE | Decode / execute the instruction |

## ASR – Arithmetic Shift Right

This instruction shifts the accumulator to the right while preserving the sign bit. The least significant bit is placed into the carry flag.

**Opcode Format (2 bytes)**

|  |  |
| --- | --- |
| 42 | 0A |

3 clock cycles

The C, N, and Z flags are updated by this instruction.

**Machine States:**

|  |  |
| --- | --- |
| IFETCH | Fetch the instruction |
| DECODE | Decode the page 2 prefix |
| DECODE | Decode / execute the instruction |

## BGT – Branch if Greater Than

This is a branch based on a signed comparison of two values. It takes only the negative and zero flags into consideration. The branch is taken if both the negative and zero flags are false. This instruction improves code density and performance compared to performing a sequence of instructions to synthesize this operation. Overflow can be checked with the BVS instruction prior to executing BGT.

**Opcode Format (3/5 bytes)**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 42 | 10 | Disp8 |  |  | short |
| 42 | 10 | FFh | Disp16 | | long |

3 clock cycles (regardless of taken or not taken).

No flags are affected by this instruction.

**Machine States:**

|  |  |
| --- | --- |
| IFETCH | Fetch the instruction |
| DECODE | Decode the page 2 prefix |
| DECODE | Decode / execute the instruction |

## BLE – Branch if Less or Equal

This is a branch based on a signed comparison of two values. Only the negative and zero flags are tested. This instruction is the same as a combination of BEQ and BMI. This instruction improves code density and performance compared to performing a sequence of instructions to synthesize this operation. If an overflow check is required the BVS instruction can be used prior to executing this instruction.

**Opcode Format (3/5 bytes)**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 42 | B0 | Disp8 |  |  |
| 42 | B0 | FFh | Disp16 | |

3 clock cycles (regardless of taken or not taken).

No flags are affected by this instruction.

**Machine States:**

|  |  |
| --- | --- |
| IFETCH | Fetch the instruction |
| DECODE | Decode the page 2 prefix |
| DECODE | Decode / execute the instruction |

## BMC – Bitmap Clear

This instruction clears the bit specified in the accumulator in a bitmap. The bitmap is a maximum of 512MB in size.

**Opcode Format (6 bytes)**

|  |  |  |  |
| --- | --- | --- | --- |
| 42 | 24 | Address32 |  |

8 clock cycles (4 + 2 memory accesses).

ZF, NF are set according to the result.

**Machine States:**

|  |  |
| --- | --- |
| IFETCH | Fetch the instruction |
| DECODE | Decode the page 2 prefix |
| DECODE | Decode / execute the instruction |
| LOAD1 | Fetch byte |
| LOAD2 |  |
| CALC |  |
| STORE1 | Store byte |
| STORE2 |  |

## BMS – Bitmap Set

This instruction sets the bit specified in the accumulator in a bitmap. The bitmap is a maximum of 512MB in size.

**Opcode Format (6 bytes)**

|  |  |  |  |
| --- | --- | --- | --- |
| 42 | 24 | Address32 |  |

8 clock cycles (4 + 2 memory accesses).

ZF, NF are set according to the result.

**Machine States:**

|  |  |
| --- | --- |
| IFETCH | Fetch the instruction |
| DECODE | Decode the page 2 prefix |
| DECODE | Decode / execute the instruction |
| LOAD1 | Fetch byte |
| LOAD2 |  |
| CALC |  |
| STORE1 | Store byte |
| STORE2 |  |

## BMT – Bitmap Test

This instruction tests the bit specified in the accumulator in a bitmap. The bitmap is a maximum of 512MB in size.

**Opcode Format (6 bytes)**

|  |  |  |  |
| --- | --- | --- | --- |
| 42 | 24 | Address32 |  |

6 clock cycles (4 + 1 memory access).

ZF, NF are set according to the result.

ZF is set false if the bit is a one, otherwise ZF is set true

NF is set true if the bit is a one and it’s bit #7 of the byte.

**Machine States:**

|  |  |
| --- | --- |
| IFETCH | Fetch the instruction |
| DECODE | Decode the page 2 prefix |
| DECODE | Decode / execute the instruction |
| LOAD1 | Fetch byte |
| LOAD2 |  |
| CALC |  |

## BYT –Byte Operation Prefix

The BYT prefix causes the memory access of the following instruction to be byte sized regardless of the settings in the status register. The BYT prefix may be specified in assembler code by appending a “.B” to the instruction mnemonic. When a register is being loaded as a result of a byte sized memory operation, the value from memory is sign extended to the size of the register.

**Opcode Format (2 bytes)**

|  |  |
| --- | --- |
| 42 | 8B |

2 clock cycles

No flags are affected by this instruction.

**Machine States:**

|  |  |
| --- | --- |
| IFETCH | Fetch the instruction |
| DECODE | Decode the page 2 prefix |
| DECODE | Decode following instruction |
| …. |  |

**Sample Assembler:**

LDA.B $1000,Y ; load and sign extend byte into sixteen bit accumulator

## CACHE

CACHE issues a command to the cache. Currently only three commands are supported:

00 – invalidate entire instruction cache, (3 clock cycles)

01 – invalidate instruction cache line identified by accumulator (3 clock cycles)

02 – preload instruction cache line identified by accumulator ( 19 clock cycles 3 + 16 memory)

When the instruction cache line needs to be identified the accumulator holds the address desired to be invalidated, not the line number. The line number is determined by the address. Currently with a 16 byte cache line size the address is shifted right four times and masked with $FF to determine the line number. The cache line is loaded using back-to-back memory read operations.

**Opcode Format (3 bytes)**

|  |  |  |
| --- | --- | --- |
| 42 | E0 | Immediate8 |

No flags are affected by this instruction.

**Machine States:**

|  |  |
| --- | --- |
| IFETCH | Fetch the instruction |
| DECODE | Decode the page 2 prefix |
| DECODE | Decode / execute the instruction |
| ICACHE2 | Only for preloads |
| … | “ repeats 15 more times |

## CLI – Clear Interrupt Mask

The CLI instruction sets the interrupt mask level to zero enabling all interrupts.

2 clock cycles

No flags other than the interrupt mask level affected by this instruction.

**Machine States:**

|  |  |
| --- | --- |
| IFETCH | Fetch the instruction |
| DECODE | Decode / execute |

## CMC – Compliment Carry

This instruction complements the carry flag. While not used very often, it can be tricky to complement the carry flag. Availability of this instruction eases some programming tasks.

**Opcode Format (2 bytes)**

|  |  |
| --- | --- |
| 42 | 18 |

3 clock cycles

The carry flag is inverted.

**Machine States:**

|  |  |
| --- | --- |
| IFETCH | Fetch the instruction |
| DECODE | Decode the page 2 prefix |
| DECODE | Decode / execute the instruction |

## DEX4 – Decrement .X by Four

Decrement the .X index register by four. This instruction is similar to the DEX instruction except that it decrements by four rather than by one. With a 32 bit word size for most registers arrays are often 32 bits (four bytes). Indexing into word arrays requires adjusting the index by four.

**Opcode Format (2 bytes)**

|  |  |
| --- | --- |
| 42 | CA |

3 clock cycles

N and Z flags are affected.

**Machine States:**

|  |  |
| --- | --- |
| IFETCH | Fetch the instruction |
| DECODE | Decode the page 2 prefix |
| DECODE | Decode / execute the instruction |

## DEY4 – Decrement .Y by Four

Decrement the .Y index register by four. This instructions is similar to the DEY instruction except that it decrements by four rather than by one. With a 32 bit word size for most registers arrays are often 32 bits (four bytes). Indexing into word arrays requires adjusting the index by four.

**Opcode Format (2 bytes)**

|  |  |
| --- | --- |
| 42 | 88 |

3 clock cycles

N and Z flags are affected.

**Machine States:**

|  |  |
| --- | --- |
| IFETCH | Fetch the instruction |
| DECODE | Decode the page 2 prefix |
| DECODE | Decode / execute the instruction |

## FIL

Fill memory. This instruction fills a block of memory with the value contained in the .X register. The .Y register along with the data bank specified in the instruction points to the beginning of the block of memory. The .Y register is incremented and the accumulator decremented until the accumulator reaches minus one. The accumulator holds one less than the count of how many bytes to fill. In 8/16 modes, at the end of the fill operation the data bank is loaded with the value specified in the instruction. The data bank setting is ignored in 32 bit mode. The MVN / MVP instructions may also be used to fill a block of memory however the FILL instruction is faster as it only performs memory stores.

FILL normally fills with bytes however it may be prefixed with a size code in order to cause the fill to work with either half-words (16 bits) or words (32 bits). For example the assembler syntax for a word oriented fill is: FILL.W. Using a size code prefix causes the fill operation to take two more clock cycles per unit filled.

In native 32 bit mode the data bank specified in the instruction is not used to determine the fill address. The block filled may span a bank boundary. The instruction remains three bytes long.

A segment override prefix may be applied to this instruction however the prefix will cause the clock cycle count to increase by two per each byte stored. It may be faster to save before the instruction and restore the data segment afterwards. Note that attempts to fill the code segment (using the CS: prefix) will be ignored.

The fill operation is interruptible.

This mnemonic is closely resembles the FILL pseudo-op, as a memory aid as to which is which instruction mnemonics are usually three characters long. The FILL pseudo-op statically fills a region of memory at time of assembly. The FIL instruction fills memory dynamically at run-time.

**Opcode Format (3 bytes)**

|  |  |  |
| --- | --- | --- |
| 42 | 44 | DBR8 |

6 clock cycles per byte stored (4 + 2 memory accesses)

No flags are affected by this instruction.

**Machine States:**

The following machine states repeat until the store is complete.

|  |  |
| --- | --- |
| IFETCH | Fetch the instruction |
| DECODE | Decode the page 2 prefix |
| DECODE | Decode / execute the instruction |
| STORE1 |  |
| STORE2 | store x[7:0] and Increment / decrement registers |
| STORE21 | store x[15:8] and Increment / decrement registers |
| STORE22 | store x[23:16] and Increment / decrement registers |
| STORE22 | store x[31:24] and Increment / decrement registers |
| MVN816 | Test accumulator for -1 |

1 this cycle is present only for half-word and word fill operations.

2 this cycle is present only for word fill operations.

## INF - Information

The INF instruction can be used to return general information about the processor including the contents of task registers.

Bits 4 to 15 of the .X index register indicate which context to return information for. Bits 0 to 3 of the .X index register indicate which field of information to return.

|  |  |  |
| --- | --- | --- |
| X[3:0] | Information returned |  |
| 0 | CS register |  |
| 1 | DS register |  |
| 2 | Program counter and program bank |  |
| 3 | accumulator |  |
| 4 | .X index register |  |
| 5 | .Y index register |  |
| 6 | SP – stack pointer |  |
| 7 | SR,SRX – status register and extended status register |  |
| 8 | DBR – data bank register |  |
| 9 | DPR – direct page register |  |
| 10 | back link |  |
| 11 |  |  |
| 12 |  |  |
| 13 |  |  |
| 14 |  |  |
| 15 |  |  |

If bits 4 to 15 of the .X register are all ones, then INF returns global information about the core.

**Global Information Returned:**

|  |  |  |
| --- | --- | --- |
| X[3:0] | Information returned |  |
| 0 | core number |  |
| 1 | core version |  |
| 2 |  |  |
| 3 |  |  |
| 4 |  |  |
| 5 |  |  |
| 6 |  |  |
| 7 |  |  |
| 8 |  |  |
| 9 |  |  |
| 10 |  |  |
| 11 |  |  |
| 12 |  |  |
| 13 |  |  |
| 14 |  |  |
| 15 |  |  |

**Opcode Format (2 bytes)**

|  |  |
| --- | --- |
| 42 | 4A |

4 clock cycles

N and Z flags are affected.

**Machine States:**

|  |  |
| --- | --- |
| IFETCH | Fetch the instruction |
| DECODE | Decode the page 2 prefix |
| DECODE | Decode / select context reg |
| INF1 | obtain info, select original context reg |

## INX4 – Increment .X by Four

Increment the .X index register by four. This instruction is similar to the INX instruction except that it increments by four rather than by one. With a 32 bit word size for most registers arrays are often 32 bits (four bytes). Indexing into word arrays requires adjusting the index by four.

**Opcode Format (2 bytes)**

|  |  |
| --- | --- |
| 42 | E8 |

3 clock cycles

N and Z flags are affected.

**Machine States:**

|  |  |
| --- | --- |
| IFETCH | Fetch the instruction |
| DECODE | Decode the page 2 prefix |
| DECODE | Decode / execute the instruction |

## INY4 – Increment .Y by Four

Increment the .Y index register by four. This instruction is similar to the INY instruction except it increments by four rather than by one. With a 32 bit word size for most registers arrays are often 32 bits (four bytes). Indexing into word arrays requires adjusting the index by four.

**Opcode Format (2 bytes)**

|  |  |
| --- | --- |
| 42 | C8 |

3 clock cycles

N and Z flags are affected.

**Machine States:**

|  |  |
| --- | --- |
| IFETCH | Fetch the instruction |
| DECODE | Decode the page 2 prefix |
| DECODE | Decode / execute the instruction |

## LDO – Load Offset Register

The LDO instruction loads the program offset latch register and set the OS flag bit in the status register. The offset latch register will be copied to the program offset register the next time program flow changes. At that point the OS flag bit will also be cleared.

**Opcode Format (2 bytes)**

|  |  |  |
| --- | --- | --- |
| 42 | A2 | Immediate32 |

3 clock cycles

The OS flag bit is set.

**Machine States:**

|  |  |
| --- | --- |
| IFETCH | Fetch the instruction |
| DECODE | Decode the page 2 prefix |
| DECODE | Decode / execute the instruction |

## MUL - Multiply

MUL – Performs an unsigned multiply of the .A and .X registers and leaves the product in the accumulator and .X register. When multiplying byte registers the 16 bit product is available in the .A (low order) and .B (higher order) registers. Multiply respects the register size settings. Higher order product bits are available with the XBA and [XBAW](#_XBAW) instruction when operating in 8/16 bit mode.

Bits 0 to 31 of the product are placed in the accumulator.

Bits 32 to 63 of the product are placed into the .X register.

**Opcode Format (2 bytes)**

|  |  |
| --- | --- |
| 42 | 2A |

3 clock cycles

The N flag is set to bit 31 of the result. The Z flag is set if the result is zero. The V flag is set if the high order 32 bits of the product are non-zero.

**Machine States:**

|  |  |
| --- | --- |
| IFETCH | Fetch the instruction |
| DECODE | Decode the page 2 prefix |
| DECODE | Decode / execute the instruction |

## PHO – Push Offset Register

PHO – pushes the offset register onto an internal stack

**Opcode Format (2 bytes)**

|  |  |
| --- | --- |
| 42 | 0B |

3 clock cycles

No flags are affected by this instruction.

**Machine States:**

|  |  |
| --- | --- |
| IFETCH | Fetch the instruction |
| DECODE | Decode the page 2 prefix |
| DECODE | Decode / Execute the instruction |

## PLO – Pull Offset Register

PHO – pulls the offset register from an internal stack

**Opcode Format (2 bytes)**

|  |  |
| --- | --- |
| 42 | 2B |

3 clock cycles

No flags are affected by this instruction.

**Machine States:**

|  |  |
| --- | --- |
| IFETCH | Fetch the instruction |
| DECODE | Decode the page 2 prefix |
| DECODE | Decode / Execute the instruction |

## RTI - Return From Interrupt

The operation of this instruction has been modified. The offset register is pulled from the internal stack.

**Opcode Format (1 bytes)**

|  |
| --- |
| 40 |

12 clock cycles 2 + 5 memory accesses

No flags are affected by this instruction.

**Machine States (task return):**

|  |  |
| --- | --- |
| IFETCH | Fetch the instruction |
| DECODE | Decode / execute |
| LOAD1/2 | load SR[7:0] |
| LOAD1/2 | load SR[15:8] |
| LOAD1/2 | load PC[7:0] |
| LOAD1/2 | load PC[15:8] |
| LOAD1/2 | load PC[23:16] |

## RTL – Long Return From Subroutine

This is an additional form for the existing RTL instruction. The RTL instruction performs a long return from subroutine operation. A twenty-four bit value is popped from the stack and placed into the program counter and program bank registers. In addition the stack pointer may be incremented by an amount specified by the instruction in order to pop arguments off the stack.

**Opcode Format (3 bytes)**

|  |  |  |
| --- | --- | --- |
| 42 | 68 | Immed8 |

10 clock cycles (4 + 3 memory accesses)

No flags are affected by this instruction.

**Machine States:**

|  |  |
| --- | --- |
| IFETCH | Fetch the instruction |
| DECODE | Decode page 2 prefix |
| DECODE | Decode / execute –save register set |
| LOAD1/2 | load PC[7:0] |
| LOAD1/2 | load PC[15:8] |
| LOAD1/2 | load PC[23:16] |
| RTS1 | increment PC |

## RTS – Return From Subroutine

This is an additional form for the existing RTS instruction. The RTS instruction performs a short return from subroutine operation. A sixteen bit value is popped from the stack and placed into the program counter. The program bank is not affected. In addition the stack pointer may be incremented by an amount specified by the instruction in order to pop arguments off the stack.

**Opcode Format (3 bytes)**

|  |  |  |
| --- | --- | --- |
| 42 | C0 | Immed8 |

8 clock cycles (4 + 2 memory accesses)

No flags are affected by this instruction.

**Machine States:**

|  |  |
| --- | --- |
| IFETCH | Fetch the instruction |
| DECODE | Decode page 2 prefix |
| DECODE | Decode / execute –save register set |
| LOAD1/2 | load PC[7:0] |
| LOAD1/2 | load PC[15:8] |
| RTS1 | increment PC |

## SEI – Set Interrupt Mask Level

The SEI instruction has a new immediate addressing mode which allows it to set the core’s interrupt mask level. When the interrupt mask level is set interrupts are not globally disabled, instead they remain enabled. However only interrupts at a higher level than the mask setting may occur.

If SEI is used without an interrupt level specified then the mask level is set to seven disabling all interrupts. Valid values of the mask level are 0 to 3.

**Opcode Format (3 bytes)**

|  |  |  |
| --- | --- | --- |
| 42 | 78 | Level |

3 clock cycles

No flags other than the interrupt mask level affected by this instruction.

**Machine States:**

|  |  |
| --- | --- |
| IFETCH | Fetch the instruction |
| DECODE | Decode / execute the prefix |
| DECODE | execute level setting |

## TAO – Transfer .A to Program Offset

This instruction transfers the 32 bit accumulator to the program offset latch register and sets the offset changed bit (OS) in the status register. The latch register will be copied to the offset register on the next flow control operation (JMP, JSR, JML, JSL, branches).

**Opcode Format (2 bytes)**

|  |  |
| --- | --- |
| 42 | 1B |

3 clock cycles

The OS flag is set.

**Machine States:**

|  |  |
| --- | --- |
| IFETCH | Fetch the instruction |
| DECODE | Decode page 2 prefix |
| DECODE | Decode / execute |

## TOA – Transfer Program Offset to .A

This instruction transfers the 32 bit program offset register to the accumulator and clears the offset changed bit (OS) in the status register.

**Opcode Format (2 bytes)**

|  |  |
| --- | --- |
| 42 | 3B |

3 clock cycles

The OS flag is cleared.

**Machine States:**

|  |  |
| --- | --- |
| IFETCH | Fetch the instruction |
| DECODE | Decode page 2 prefix |
| DECODE | Decode / execute |

## XBAW – Exchange B and A Words

Exchange high order and low order word of accumulator. Bits 0 to 15 are exchanged with bits 16 to 31. Operation of this instruction is similar to the XBA instruction. This instruction can be used to obtain access to bits 16 to 31 of the multiplier product. This instruction combined with the XBA instruction can be used to switch the byte order around.

**Opcode Format (2 bytes)**

|  |  |
| --- | --- |
| 42 | EB |

3 clock cycles

N is set to bit 15 of the result. Z is set if bits 0 to 15 of the result are zero.

**Machine States:**

|  |  |
| --- | --- |
| IFETCH | Fetch the instruction |
| DECODE | Decode the page 2 opcode |
| DECODE | Decode and execute the instruction |

# Core Parameters

|  |  |  |
| --- | --- | --- |
| Parameter | Default value | What it does |
| EXTRA\_LONG\_BRANCHES | 1 | Causes the core to generate hardware to support extra-long branching for the general purpose branch instructions. |
| PC24 | 1 | Causes the program counter to be a true 24 bit program counter (increments automatically across banks). Set to zero to force a 16 bit program counter which wraps around at a bank boundary. Setting this value to zero may generate slightly less hardware and is consistent with the 65c816. |
| POPBF | 0 | If set to one, allows popping the break flag from the stack. The default setting is consistent with 65xxx operation. |

# Configuration Defines

|  |  |  |
| --- | --- | --- |
|  | Default Value | What it does |
| ICACHE\_4K | 1 | Causes the core to use a 4kB instruction cache. |
| ICACHE\_16K | 0 | Causes the core to use a 16kB instruction cache. Cannot be defined at the same time as ICACHE\_4K. |
| SUPPORT\_BCD | 1 | Causes the core to include logic to support BCD addition and subtraction. BCD support is necessary to remain compatible with the 65xxx series. |
| SUPPORT\_NEW\_INSN | 1 | Causes the core to include new instructions. Commenting out this definition will significantly reduce the size of the core; however instructions supporting new core features will not be available. |

# I/O Ports

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | In/Out | Width |  |  |
| corenum | I | 32 | core number, if left unassigned zero is assumed. This input is reflected by the INF instruction. |  |
| rst | I | 1 | reset, active low – resets the core |  |
| clk | I | 1 | input clock, this clock is not directly used to clock the core. Instead it is gated internally to allow the core clock to be stopped with the STP instruction. |  |
| clko | O | 1 | output clock. – this is the input clock gated and drives the core. this clock may stop if the STP instruction is executed. |  |
| phi11 | O | 1 | Phase one of the input clock divided by 32. This is a low speed clock output designed to drive peripherals. |  |
| phi12 | O | 1 | Phase two of the input clock divided by 32. This is a low speed clock output designed to drive peripherals. |  |
| phi81 | O | 1 | Phase one of the input clock divided by 8. This is a low speed clock output designed to drive peripherals / low speed memory. |  |
| phi82 | O | 1 | Phase two of the input clock divided by 8. This is a low speed clock output designed to drive peripherals / low speed memory. |  |
| nmi | I | 1 | active low input for non-maskable interrupt |  |
| irq | I | 1 | active low input for interrupt |  |
| abort | I | 1 | active low input for abort interrupt |  |
| e | O | 1 | ‘e’ flag indicator reflects the status of the emulation flag |  |
| mx | O | 1 | m and x status output ‘m’ when clock is high, otherwise ‘x’ |  |
| rdy | I | 1 | active high ready input, pull low to insert wait states |  |
| be | I | 1 | bus enable, tri-states the address, data, and r/w lines when active |  |
| vpa | O | 1 | valid program address, set high during an instruction cache line fetch |  |
| vda | O | 1 | valid data address, set high during a data access, also set high during the first cycle of an instruction cache line fetch |  |
| mlb | O | 1 | memory lock, active high |  |
| vpb | O | 1 | vector pull, set high during a vector fetch |  |
| rw | O | 1 | read/write, active high for read, low for write cycle |  |
| ad | O | 32 | address bus |  |
| db | I/O | 8 | data bus , input for read cycles, output for write cycles |  |
|  |  |  |  |  |

# Opcode Map

Opcode Map – 8 bit mode W65C816 compatible

|  |  |
| --- | --- |
|  | = W65C816S instructions |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | -0 | -1 | -2 | -3 | -4 | -5 | -6 | -7 | -8 | -9 | -A | -B | -C | -D | -E | -F |
| 0- | BRK | ORA (d,x) | COP | ORA d,s | TSB d | ORA d | ASL d | ORA [d] | PHP | OR #i8 | ASL acc | PHD | TSB abs | ORA abs | ASL abs | ORA AL |
| 1- | BPL disp | ORA (d),y | ORA (d) | ORA (d,s),y | TRB d | OR d,x | ASL d,x | ORA [d],y | CLC | OR abs,y | INA | TAS | TRB abs | ORA abs,x | ASL abs,x | ORA AL,x |
| 2- | JSR abs | AND (d,x) | JSL abs24 | AND d,s | BIT d | AND d | ROL d | AND [d] | PLP | AND #i8 | ROL acc | PLD | BIT abs | AND abs | ROL abs | AND AL |
| 3- | BMI disp | AND (d),y | AND (d) | AND (d,s),y | BIT d,x | AND d,x | ROL d,x | AND [d],y | SEC | AND abs,y | DEA | TSA | BIT abs,x | AND abs,x | ROL abs,x | AND AL,x |
| 4- | RTI | EOR (d,x) | WDM | EOR d,s | MVP | EOR d | LSR d | EOR [d] | PHA | EOR #i8 | LSR acc | PHK | JMP abs | EOR abs | LSR abs | EOR AL |
| 5- | BVC disp | EOR (d),y | EOR (d) | EOR (d,s),y | MVN | EOR d,x | LSR d,x | EOR [d],y | CLI | EOR abs,y | PHY | TCD | JML abs24 | EOR abs,x | LSR abs,x | EOR AL,x |
| 6- | RTS | ADC (d,x) | PER | ADC d,s | STZ d | ADC d | ROR d | ADC [d] | PLA | ADC #i8 | ROR acc | RTL | JMP (abs) | ADC abs | ROR abs | ADC AL |
| 7- | BVS disp | ADC (d),y | ADC (d) | ADC (d,s),y | STZ d,x | ADC d,x | ROR d,x | ADC [d],y | SEI | ADC abs,y | PLY | TDC | JMP (abs,x) | ADC abs,x | ROR abs,x | ADC AL,x |
| 8- | BRA disp | STA (d,x) | BRL disp | STA d,s | STY d | STA d | STX d | STA [d] | DEY | BIT # | TXA | PHB | STY abs | STA abs | STX abs | STA AL |
| 9- | BCC disp | STA (d),y | STA (d) | STA (d,s),y | STY d,x | STA d,x | STX d,y | STA [d],y | TYA | STA abs,y | TXS | TXY | STZ abs | STA abs,x | STZ abs,x | STA AL,x |
| A- | LDY #i8 | LDA (d,x) | LDX #i8 | LDA d,s | LDY d | LDA d | LDX d | LDA [d] | TAY | LDA #i8 | TAX | PLB | LDY abs | LDA abs | LDX abs | LDA AL |
| B- | BCS disp | LDA (d),y | LDA (d) | LDA (d,s),y | LDY d,x | LDA d,x | LDX d,y | LDA [d],y | CLV | LDA abs,y | TSX | TYX | LDY abs,x | LDA abs,x | LDX abs,x | LDA AL,x |
| C- | CPY #i8 | CMP (d,x) | REP # | CMP d,s | CPY d | CMP d | DEC d | CMP [d] | INY | CMP #i8 | DEX | WAI | CPY abs | CMP abs | DEC abs | CMP AL |
| D- | BNE disp | CMP (d),y | CMP (d) | CMP (d,s),y | PEI | CMP d,x | DEC d,r | CMP [d],y | CLD | CMP abs,y | PHX | STP | JML (a) | CMP abs,x | DEC abs,x | CMP AL,x |
| E- | CPX #i8 | SBC(d,x) | SEP # | SBC d,s | CPX d | SUB d | INC d | SBC [d] | INX | SBC #i8 | NOP | XBA | CPX abs | SBC abs | INC abs | SBC AL, |
| F- | BEQ disp | SBC (d),y | SBC(r) | SBC (d,s),y | PEA | SUB d,x | INC d,r | SBC [d],y | SED | SBC abs,y | PLX | XCE | JSR (abs,x) | SBC abs,x | INC abs,x | SBC AL,x |

Opcode Map – Page 2 Opcodes

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | -0 | -1 | -2 | -3 | -4 | -5 | -6 | -7 | -8 | -9 | -A | -B | -C | -D | -E | -F |
| 0- | BRK2 | ORA {d,x} |  |  |  |  |  |  |  |  | ASR acc | PHO | TSB xlabs | ORA xlabs | ASL xlabs |  |
| 1- | BGT disp | ORA {d},y | ORA {d} | ORA {d,s},y | BMT xlabs |  |  |  | CMC | OR xlabs,y | TTA | TAO | TRB xlabs | ORA xlabs,x | ASL xlabs,x |  |
| 2- |  | AND {d,x} | JSF seg:offs |  | BMS xlabx |  |  |  |  |  | MUL | PLO | BIT xlabs | AND xlabs | ROL xlabs |  |
| 3- |  | AND {d},y | AND {d} | AND {d,s},y | BMC xlabs |  |  |  |  | AND xlabs,y |  | TOA | BIT xlabs,x | AND xlabs,x | ROL xlabs,x |  |
| 4- |  | EOR {d,x} | WDM2 |  | FIL |  |  |  |  |  | INF |  | LDT xlabs,x | EOR xlabs | LSR xlabs |  |
| 5- |  | EOR {d},y | EOR {d} | EOR {d,s},y |  |  |  |  |  | EOR xlabs,y |  | : | JMF seg:offs | EOR xlabs,x | LSR xlabs,x |  |
| 6- |  | ADC {d,x} | JCF |  |  |  |  |  | RTL # |  |  |  | LDT xlabs | ADC xlabs | ROR xlabs |  |
| 7- |  | ADC {d},y | ADC {d} | ADC {d,s},y |  |  |  |  | SEI # | ADC xlabs,y |  | : | JML [xlabs,x] | ADC xlabs,x | ROR xlabs,x |  |
| 8- |  | STA {d,x} | JCL |  |  |  |  |  | DEY4 |  | AAX | BYT: | STY xlabs | STA xlabs | STX xlabs |  |
| 9- |  | STA {d},y | STA {d} | STA {d,s},y |  |  |  |  |  | STA xlabs,y | WRD: | UBT: | STZ xlabs | STA xlabs,x | STZ xlabs,x |  |
| A- |  | LDA {d,x} | LDO # |  |  |  |  |  |  |  |  | HAF: | LDY xlabs | LDA xlabs | LDX xlabs |  |
| B- | BLE disp | LDA {d},y | LDA {d} | LDA {d,s},y |  |  |  |  |  | LDA xlabs,y |  | UHF: | LDY xlabs,x | LDA xlabs,x | LDX xlabs,x |  |
| C- | RTS # | CMP {d,x} | REP # |  |  |  |  |  | INY4 |  | DEX4 |  | CPY xlabs | CMP xlabs | DEC xlabs |  |
| D- |  | CMP {d},y | CMP {d} | CMP {d,s},y | PEA { } |  |  |  |  | CMP xlabs,y |  | CLK | JML [xlabs] | CMP xlabs,x | DEC xlabs,x |  |
| E- | CACHE # | SBC{d,x} | SEP # |  |  |  |  |  | INX4 | INC z,# | NOP2 | XBAW | CPX xlabs | SBC xlabs | INC xlabs |  |
| F- | PCHIST | SBC {d},y | SBC{d} | SBC {d,s},y | PEA xlabs |  |  |  |  | SBC xlabs,y |  |  | JSL [xlabs,x] | SBC xlabs,x | INC xlabs,x |  |