s80x86 Core Development Guide

Jamie Iles

Table of Contents

Developer’s Guide

Building

System Requirements

- Docker, tested with version 1.12.1.

Quick Start Build

The `scripts/build` script provides everything that is needed to quickly build and test the project. On the first run, the script will build the required Docker images from `docker/build/Dockerfile`, configure and build the project and run all of the built-in tests.

```
./scripts/build
```

Build Environments

The s80x86 project uses Docker to provide convenient build environments. This means that it is possible to build and test the design on any Linux system with Docker, regardless of distribution. The Docker images used internally are all based on Ubuntu 16.04 LTS.

There are two primary build environments:

- s80x86-build
- s80x86-dev

Each build environment has a script in `docker` to enter the container and build it if not already built. A 'ccache' cache is created in `_build/.ccache` that persists across container runs to increase build performance.

's80x86-build' is a minimal Ubuntu 16.04 LTS container with the dependencies for building the project and running the tests. This environment runs everything as the current user’s uid/gid to preserve file permissions outside of the container and bind-mounts the project directory into `/build`. The 's80x86-dev' container is intended for developing the project itself and includes extra packages and convenience scripts to make developing easier. Unlike the 's80x86-build' container, this container bind mounts `/home` from the host into the container for convenience and includes packages like GTKWave for viewing waveforms. It is recommended to use the Docker build environments for all builds and developments as those are used for the baseline development and verification.

CI Build Scripts

Several build scripts are included suitable for use in a continuous integration environment.

scripts/ci/unittest builds the project from scratch and then runs all of the unit tests, producing JUnit XML suitable for importing into the CI test runner history.

scripts/ci/gcov-coverage performs the same steps as `scripts/ci/coverage` but produces Cobertura compatible coverage information that can be read into Jenkins or other CI systems supporting this format.

Build Configurations

The CMake based build system supports the following build configurations and can be selected by passing `-DCMAKE_BUILD_TYPE=’CONFIGURATION' to `cmake`.

Release is optimized for performance, no debug information.

Debug enables debug information in all C/C++ executables, and tracing of Verilog models which will write VCD files for each test run.

Coverage builds everything for coverage including C++ and Verilog.

Configuration Options

There are some CMake configuration options that control features:

- **-DS80X86_TRAP_ESCAPE** controls whether the escape fault is taken on an escape opcode. This defaults to off for DOS compatibility in which case escape opcodes act like a NOP.

Simulator

The simulator can run either the software simulation (SoftwareCPU) or the RTL model (RTLCPU). Typing `^]` will cause the simulator to exit.

```
Options:
  -h [ --help ]         print this usage information and exit
  -b [ --backend ] arg  the simulator backend module to use, either
SoftwareCPU
                        or RTLCPU, default SoftwareCPU
  -r [ --restore ] arg  restore file to load from
  -s [ --save ] arg     save file to write from
  --bios arg            the bios image to use
  --diskimage arg       the boot disk image
```

The save and restore functionality are a useful tool for debugging - the entire simulation state can be saved at exit and restored later making it possible to checkpoint once the system is in a known state and then repeatedly debug without waiting for the system to return to the same state. Importantly, this is transferable between backends, so it is possible to boot the system and run an application with the `SoftwareCPU` and then exit and restart with the `RTLCPU` to greatly reduce time taken to get to the interesting debug point.

Microarchitecture

The core has a loosely coupled three stage pipeline consisting of instruction fetch, instruction decode and execution. The prefetch stage has an 8 byte queue which feeds into a hardware instruction decoder that fills a 4 deep fully decoded instruction queue.

Deviating from the 80186, the hardware decoder generates an invalid opcode trap when an instruction exceeds 15 bytes in length. This is not a valid 16-bit encoding and can only be reached with multiple redundant prefixes.

Microcode

The microcode is stored in `rtl/microcode` where microcode files have the `.us` suffix. The microassembler first passes these files through the C preprocessor to allow inclusion of other files and creating macros.

Directives

Directives are used to provide information to the microassembler about microcode layout without actually generating microinstructions.

Table 1. Microassembler Directives

Name	Description
.opcode NUM	The `.opcode NUM` directive tells the microassembler to insert the next microinstruction at address `NUM` in the microprogram. This is used for the first 256 opcodes so that efficient dispatch can be performed by jumping to the address corresponding to the value of the opcode.
.auto_address	Returns the address assignment to automatically assigned addresses after using the `.opcode` directive.

Microinstruction Fields

Jumps

Table 2. Microcode Jump Types

Name	Description

jmp_rm_reg_mem LABEL	tells the microassembler to generate a jump that will jump to the label if the Mod R/M decoder indicates a register operand in the R/M field and the label + 1 if the R/M field encodes a memory operand. For example:
	```
	    jmp_rm_reg_mem foo_reg;
	foo_reg:
	    next_instruction;
	foo_mem:
	    next_instruction;
	```
	 will jump to `foo_reg` if the R/M operand is a register operand and `foo_mem` if the R/M operand is a memory operand. The two microinstructions must be adjacent with the register based instruction appearing first.

jmp_opcode	Takes the value of the opcode that was fetched and jumps to that address as an absolute value, used in combination with the `.at` directive to implement opcodes.
jmp_dispatch_reg LABEL	Using the reg field of the mod r/m byte, jump to the target address + the value of the reg field. Used for implementing different instructions that share the same opcode.

jmp_if_not_rep LABEL	Jump to the target address if the string instruction does not have a rep prefix, otherwise continue execution at the next incremented address. Only valid on string instructions that may be combined with a rep prefix.
jmp_if_zero LABEL	Jump to the target address if the Z flag is set, otherwise continue with the adjacent instruction. Note that this uses the flags register and not the combinational flags output of the current ALU operation.
jmp_rb_zero LABEL	Jump to the target address if RB value is zero, otherwise continue with the adjacent instruction.
jmp_if_rep_not_taken LABEL	Check the condition for the current rep prefix and jump to the target if the termination condition is not met, otherwise execute the adjacent instruction. Only valid when there is a rep prefix present.
jmp_if_taken LABEL	Jump to the target if the jump instruction has the condition met, otherwise continue with the adjacent instruction. This is only valid for jump instructions, and INTO.
jmp_loop_done LABEL	Jump to the target if loop has completed. The loop counter is loaded from the 5 MSB’s of the immediate and decremented on each iteration. This is only used for the `enter` instruction.
jmp LABEL	An unconditional jump, will always transfer control to LABEL.

Data Sources

Table 3. Microcode Data Sources

Name	Description
ra_sel	Which general purpose register to fetch for RA. Note that register fetches have a single cycle latency. Only valid when `ra_modrm_rm_reg` is not set.
rb_cl	Set to use the value of `CL` for RB after a single cycle of latency, used primarily for shifts.
segment	Set the default segment for the memory operation or segment register read. This is the default segment and may be overriden with a segment override prefix unless `segment_force` is also set.
a_sel	Selects which operand source to use for the internal A bus: • RA: the fetched RA GPR value. • IP: the instruction pointer of the next instruction. • MAR: the contents of the memory address register. • MDR: the contents of the memory data register.

b_sel	Selects which operand source to use for the internal B bus: • RB: the fetched RB GPR value. • IMMEDIATE: an immediate value, either from the immediate reader or from the constant pool if a microinstruction defined constant is being used. • SR: the fetched segment register value. • TEMP: the contents of the temporary register.
immediate	The immediate constant to use. This forms a constant pool in the microcode and can be used for operations such as fetching exception handler addresses, incrementing/decrementing pointers etc.
mar_wr_sel	Selects the source of the value to be written to the memory address register: • EA: the effective address calculated by the mod r/m decoder. • Q: the Q bus driven by the ALU.

Control Signals

Table 4. Microcode Control Signals

Name	Description
next_instruction	Ends processing of the current instruction, will check for pending interrupts, jump to the instruction dispatch address, update CS:IP and reset any intermediate state.
mar_write	Write the value of the `mar_wr_sel` source into the memory address register.
mdr_write	Write the value of the ALU output into the memory data register.
mem_read	Perform a memory access with the specified segment and memory address register value, reading into the memory data register. Note that the segment register must have had the fetch initiated in the previous instruction and should be held for the duration of the access. This field will cause the microsequencer to stall until the access is complete. The `width` field will specify the size of the access.
mem_write	Perform a memory write, writing the contents of the memory data register to the address specified by the fetched segment and the memory address register. As with `mem_read`, the segment must have had the fetch initiated in the previous instruction and held for the duration of this instruction.
segment_force	When used in combination with the `segment` field, this will force that segment to be used unconditionally, ignoring any segment override prefix.
alu_op	The ALU operation to execute, see "scripts/microassembler/microasm/types.py" for a full list of operations.

update_flags	A list of flags that should be written when performing an ALU operation. If not specified, no flags will be update. For example: `alu_op ADD, update_flags CF OF ZF AF` will update the carry, overflow, zero and adjust flags to the result of the ALU operation.
rd_sel_source	The source of the destination register number: • MODRM_REG: use the reg field of the mod r/m byte as the destination register. • MODRM_RM_REG: use the rm field of the mod r/m byte as the destination register. • MICROCODE_RD_SEL: use the rd_sel field of the instruction to select the destination register.
reg_wr_source	Selects which result should be written to the destination register: • Q: the result of the ALU operation. • QUOTIENT: the quotient of a division operation. • REMAINDER: the remainder of a division operation.
tmp_wr_en	Set to write the output of the ALU into the temporary register.
tmp_wr_sel	Select the source of the temporary register write: • Q_LOW: the low 16-bits of the ALU output by default. • Q_HIGH: the high 16-bits of the ALU output, only used for 16x16 multiplications.
width	Selects the width of the operation. Defaults to 16-bit, but "width W8" will perform byte operations for register read/write, memory read/write, immediate fetch and ALU operations. "width WAUTO" will infer the width from the opcode, where bit 0 being set indicates a 16-bit operation.
load_ip	Causes the ALU result to be used as the new IP to be taken when the next instruction is executed.
io	Combined with `mem_read`/`mem_write` to indicate that the operation should use the I/O address space. This will cause the segment to be ignored and the io pin to be asserted for the duration of this microinstruction.
ext_int_inhibit	Used at the end of a microprogram, this flag indicates that the microsequencer should not check for interrupts after this instruction. This is used for instructions like `mov ss, bx` where the following instruction would set `sp`.
ext_int_inhibit	Used in string instructions to indicate that interrupts may be serviced at this point.

Debug

The microsequencer provides a very simple way to implement on-chip debug. The core has a number of signals to interface between a debug controller (typically JTAG) and the microsequencer. These signals are all in the core clock domain and will require synchronization with a debug controller in a different clock domain.

The debug mechanism works by putting the core into a halt mode where it will perform a tight loop in the microsequencer at which point other debug operations can be issued. Operations are issued by running a microprogram at a known address allowing more debug procedures to be added easily. To perform a debug operation, the debug controller first halts the core by raising `debug_seize` and waits for the core to enter the halted state with `debug_stopped` asserted which will be at the end of the current microprogram. Once stopped, the controller can write data to the temporary register if required with `debug_wr_val` and `debug_wr_en` and then run the debug procedure by writing the procedure address to `debug_addr` and asserting `debug_run` for a single clock cycle.

Debug Signals

Table 5. Debug Interface Signals

Name	Width	Direction	Description
debug_stopped	1	output	Asserted when the core is in a debug halt and is ready for debug operations. The debug controller must not issue any operations when `debug_stopped` is not asserted.
debug_seize	1	input	Asserted by the controller to request that the core enters debug mode. This may be deasserted once `debug_stopped` has been asserted and then the run procedure executed to continue normal operation.
debug_addr	8	input	The address of the debug procedure to execute, must be written at the same time as `debug_run`. The core will run the procedure at 100h + `debug_addr`.
debug_run	1	input	Asserted by the debug controller to begin the debug procedure specified in `debug_addr`.
debug_wr_val	16	input	Asserted by the debug controller to write the value in `debug_wr_val` into the temporary register.
debug_wr_en	1	input	Asserted by the debug controller to write `debug_wr_val` into the temporary register.

Control and Reserved Debug Procedures

- **0x00**: resume execution. If `debug_seize` is held high then this will single-step one instruction, otherwise run indefinitely until seized.
- **0x01 - 0x02: reserved for internal use, execution will yield undefined behaviour.**

Data Transfer Debug Procedures

These debug procedures are used to transfer data between the debug controller and the core.

Table 6. Data Transfer Debug Procedures

Program Number	Source	Destination
0x03	`AX`	`debug_val`
0x04	`CX`	`debug_val`
0x05	`DX`	`debug_val`
0x06	`BX`	`debug_val`
0x07	`SP`	`debug_val`
0x08	`BP`	`debug_val`
0x09	`SI`	`debug_val`
0x0a	`DI`	`debug_val`
0x0b	`ES`	`debug_val`
0x0c	`CS`	`debug_val`
0x0d	`SS`	`debug_val`
0x0e	`DS`	`debug_val`
0x0f	`IP`	`debug_val`
0x10	`FLAGS`	`debug_val`
0x11	`debug_val`	`IP`
0x12	`debug_val`	`FLAGS`
0x13	`debug_val`	`AX`
0x14	`debug_val`	`CX`
0x15	`debug_val`	`DX`
0x16	`debug_val`	`BX`
0x17	`debug_val`	`SP`
0x18	`debug_val`	`BP`
0x19	`debug_val`	`SI`
0x1a	`debug_val`	`DI`
0x1b	`debug_val`	`ES`
0x1c	`debug_val`	`CS`
0x1d	`debug_val`	`SS`
0x1e	`debug_val`	`DS`
0x1f	`debug_val`	`MAR`
0x20	`debug_val`	`MDR`
0x21	mem8[DS:MAR]	`debug_val`

Program Number	Source	Destination
0x22	mem16[DS:MAR]	`debug_val`
0x23	MDR	mem8[DS:MAR]
0x24	MDR	mem16[DS:MAR]
0x25	io8[MAR]	`debug_val`
0x26	io16[MAR]	`debug_val`
0x27	MDR	io8[MAR]
0x28	MDR	io16[MAR]

NOTE All memory transfers implicitly use DS as the segment. To write outside of the current data segment, save the value of DS, write it with the new value, perform the access and then restore DS.

FPGA JTAG

The DE0-Nano and DE0-CV boards use the Altera Virtual JTAG to implement a debug bridge between the development machine and the FPGA design. This is not a compliant JTAG TAP, but provides a reference implementation of implementing a debug interface for the core.

The implementation uses a 2 bit instruction register and variable length data register.

Table 7. JTAG Instruction Register Definitions

Register	Name
2’b00	IDCODE
2’b01	STATUS_CONTROL
2’b10	DEBUG_VALUE
2’b11	RUN_PROCEDURE

IDCODE

The IDCODE register is a 32-bit register containing the device ID code. This register is read-only, values shifted in are ignored.

STATUS_CONTROL

Table 8. JTAG STATUS_CONTROL Data Register

Field	Bits	Access	Description
RUN	[0:0]	R/W	Returns the current execution state of the CPU, "1" indicates that the core is executing in normal mode. Write a "0" to enter debug mode, polling this bit until it reflects that the core has stopped. To restart the core, write a "1", and then run debug procedure "0".
RESET	[1:1]	R/W	Core reset control, write "1" to start a reset, write "0" to clear.
RESERVED	[15:2]	RO	Reserved for future use.
WRITE_ENABLE	[16]	WO	Write as "1" to write the value shifted in into the core, otherwise the shifted value will be discarded.

DEBUG_VALUE

Table 9. JTAG DEBUG_VALUE Data Register

Field	Bits	Access	Description
VALUE	[15:0]	RW	The value to be written to/read from the debug controller.

Field	Bits	Access	Description
WRITE_ENABLE	[16]	R/W	For the value shifted in, if this is set to "1", then the VALUE will be written into the debug controller, otherwise discarded. For the value shifted out, if "1", then the VALUE field is valid. When reading, this bit should be polled until it returns "1".

RUN_PROCEDURE

Table 10. JTAG RUN_PROCEDURE Data Register

Field	Bits	Access	Description
VALUE	[7:0]	WO	The debug procedure to run. This is a write-only field.

FPGA Reference Designs

Both reference designs feature 1MB of SDRAM with other peripherals mapped over the top. Additionally, a write-back cache on the frontend of the SDRAM boosts performance of external memory accesses and is transparent to the CPU. The cache features a line size of 16 bytes to allow easy filling from SDRAM with an 8 word burst and fills from the critical word first. Accesses to the BIOS and video RAM are not cached as they utilize single cycle accesses and the video RAM needs to be coherent.

DE0-Nano

To build the DE0-Nano design, configure the build with "-DBUILD_DE0_NANO=ON". The build target "de0-nano" will build the FPGA, and "de0-nano-program" will load the design into the FPGA via the Altera USB Blaster.

Table 11. DE0-Nano Memory Map

Start	End	Description
20’h0000	20’hfffff	SDRAM

Table 12. DE0-Nano IO Port Map

Address	Width (bits)	Description
16’h0020	8	PIC command register
16’h0021	8	PIC data register
16’h0040	8	PIT channel 0 data register
16’h0043	8	PIT control register
16’hffe0	8	Mouse data register: • [7:0]: read data from the mouse, write command data
16’hffe1	8	Mouse status register: • [7]: write to reset • [6:3]: reserved • [2]: transmitter busy • [1]: unread receive error • [0]: receive FIFO not empty
16’hffec	16	BIOS control register: • [0]: BIOS ROM enabled.

Address	Width (bits)	Description
16’hfff0	16	SPI control register: • [15:10]: reserved. • [9]: CS activate. • [8:0]: clock divider.
16’hfff2	16	SPI transfer register: • [15:9]: reserved. • [8]: transfer busy. • [7:0]: transfer data. Writing to this register will initiate a one byte transfer. The CPU should then poll until bit 8 is clear at which point [7:0] will contain the received data.
16’ffff6	8	IRQ test register: - [7]: write a 1 to raise NMI, 0 to clear NMI. - [6:0]: write a 1 to raise interrupt N.
16’hfffa	8	UART data register, write to transmit data, read to fetch the received data.
16’hfffb	8	UART status register: • [7:2]: reserved. • [1]: transmitter busy. • [0]: receive data ready, cleared once the data register is read.
16’hfffc	16	SDRAM configuration register: • [15:1]: reserved. • [0]: SDRAM configuration complete. The SDRAM should not be accessed until this bit is set.
16’hfffe	16	LED register, writing will set the LED registers on the board, a 1 is enabled, 0 is disabled.

DE0-CV

To build the DE0-CV design, configure the build with "-DBUILD_DE0_CV=ON". The build target "de0-cv" will build the FPGA, and "de0-cv-program" will load the design into the FPGA via the Altera USB Blaster.

Table 13. DE0-CV Memory Map

Start	End	Description
20’h0000	20’hfffff	SDRAM

Table 14. DE0-CV IO Port Map

Address	Width (bits)	Description
16’h0020	8	PIC command register
16’h0021	8	PIC data register
16’h0040	8	PIT channel 0 data register
16’h0043	8	PIT control register
16’h0060	8	PS/2 data register: • [7:0]: read the head of the FIFO, 0 if no bytes available. Write to transmit a byte, must only be written when bit 2 of the status register is clear.
16’h0061	8	PS/2 control/status register: • [7]: RX FIFO reset. • [6:3]: reserved • [2]: transmit in progress, cleared when the transmitter is idle. • [1]: a byte with a parity error was received and discarded. Cleared on read. • [0]: receive FIFO not empty.
16’hffec	16	BIOS control register: • [0]: BIOS writable.
16’hfff0	16	SPI control register: • [15:10]: reserved. • [9]: CS activate. • [8:0]: clock divider.
16’hfff2	16	SPI transfer register: • [15:9]: reserved. • [8]: transfer busy. • [7:0]: transfer data. Writing to this register will initiate a one byte transfer. The CPU should then poll until bit 8 is clear at which point [7:0] will contain the received data.
16’ffff6	8	IRQ test register: - [7]: write a 1 to raise NMI, 0 to clear NMI. - [6:0]: write a 1 to raise interrupt N.

Address	Width (bits)	Description
16’hfffa	8	UART data register, write to transmit data, read to fetch the received data.
16’hfffb	8	UART status register: • [7:2]: reserved. • [1]: transmitter busy. • [0]: receive data ready, cleared once the data register is read.
16’hfffc	16	SDRAM configuration register: • [15:1]: reserved. • [0]: SDRAM configuration complete. The SDRAM should not be accessed until this bit is set.

BIOS

The reference BIOS is a non-compliant BIOS for demonstration purposes. The bios is built with the Mentor Graphics `ia16-elf` toolchain which is installed in the standard docker images. The BIOS uses an SD card to emulate the floppy drive and will boot the first sector of whatever is written to that SD card.

The BIOS is loaded at "f000:e000", and the BIOS stack grows down from "f000:dffe". The UART is used for video and keyboard services.

RTL Tests

The RTL tests are written in C++, using Verilator to create C++ models of the Verilog. For example, given a synchronous Fifo, the Verilator model can be created using the Verilator CMake package:

```
include(Verilator)
add_library(verilator STATIC
/usr/share/verilator/include/verilated.cpp;/usr/share/verilator/include/
verilated_cov.cpp;/usr/share/verilator/include/verilated_vcd_c.cpp;/usr/
share/verilator/include/verilated_dpi.cpp)
verilate(Fifo /home/jamie/src/80x86/documentation/Fifo.v)
```

This will generate a `verilator` library containing the common Verilator support functions, run Verilator on `Fifo.v` and generate a `VFifo` library and `VFifo.h` header for inclusion in the test code. A templated wrapper 'VerilogTestbench' in `VerilogTestbench.h` provides convenient methods for resetting and clocking the device under test along with running deferred and clock edge events, tracing and coverage.

The device under test can then be encapsulated inside a class and used for writing tests with Google Test. For example, wrapping the Verilog model:

```
#include <VFifo.h>

#include "VerilogTestbench.h"

class FifoTestbench : public VerilogTestbench<VFifo> {
public:
    FifoTestbench(VFifo *dut);
    void push(uint32_t val);
    uint32_t pop();
};

FifoTestbench::FifoTestbench(VFifo *dut)
    : VerilogTestbench<VFifo>(dut)
{
    dut->wr_en = 0;
    dut->wr_data = 0LU;
    dut->rd_en = 0;
}

void FifoTestbench::push(uint32_t val)
{
    dut->wr_data = val;
    dut->wr_en = 1;
    cycle();
    dut->wr_en = 0;
}

uint32_t FifoTestbench::pop()
{
    dut->rd_en = 1;
    cycle();
    dut->rd_en = 0;

    return dut->rd_data;
}
```

Then a test can be written to exercise it:

```
TEST(Fifo, ResetClears)
{
    FifoTestbench tb;

    for (uint32_t m = 0; m < 4; ++m)
        tb.push(m);

    ASSERT_FALSE(tb.dut->empty);
    tb.reset();
    ASSERT_TRUE(tb.dut->empty);
}
```

More complex tests that have deferred events such as reading from memory can be written by adding events on positive+negative clock edges and running after a number of cycles. `tests/rtl/TestPrefetch.cpp` uses a number of these concepts. With the right abstractions it can be possible to type-parameterize these test cases to run against pure software simulations and Verilog models.

Programmer’s Reference

Interrupts

The CPU core implements the following exceptions. Traps are handled after the instruction and the saved CS:IP points to the next instruction, faults are restartable and the saved CS:IP points to the instruction that caused the fault and so can be restarted. The core correctly handles multiple prefix bytes during an interrupted string instruction.

Table 15. Exceptions

Name	Type	Number	Description
Divide-by-zero	Trap	0	Raised when division by zero occurs or the result of the division operation does not fit in the range of the destination register.
Single-step	Trap	1	Raised when `TF` is set and the core steps an instruction.
NMI	Interrupt	2	Non-maskable interrupt, raised when the `nmi` signal has a negative to positive edge.
INT	Interrupt	3	Normal interrupt, raised by the `int3` instruction.
Overflow	Trap	4	Overflow, raised by an `into` instruction if `OF` is set.
Bound	Trap	5	Bounds check, raised when the `bound` instruction detects an out-of-bounds address.
Invalid Opcode	Trap	6	Raised when an invalid opcode is executed. Invalid opcodes do not include unimplemented opcodes or undocumented opcodes. The opcodes that raise this trap are: • 8'h0f • 8'h63 • 8'h64 • 8'h65 • 8'h66 • 8'h67 • 8'hf1 • 8'hff or 8'hfe with /reg=7 • 8'h62 (bound) with a register operand
Escape	Fault	7	Escape opcode, this will always be raised on an `esc` instruction as no coprocessors are supported.

Instructions

The following tables list all of the supported instructions along with sizes and timings for each. The timings are measured using the RTL simulation under the following conditions:

- The prefetch FIFO contains the instruction and is padded with other bytes to be full.
- The instruction and data busses are connected to an arbiter that gives priority to data accesses.
- Memory accesses take a single cycle to complete.

Instructions with a MOD R/M byte require an additional cycle to calculate the effective address.

Prefix bytes add one byte to the instruction length, and for conflicting prefixes such as multiple segment overrides, the last prefix is used.

AAA

Flags updated

C	P	A	Z	S	T	I	D	O
M		M						

Encoding

Table 16. AAA encoding

Mnemonic	Opcode	Size	Latency
`aaa AX`	`37`	1	2

AAD

Flags updated

C	P	A	Z	S	T	I	D	O
	M		M	M				

Encoding

Table 17. AAD encoding

Mnemonic	Opcode	Size	Latency
`aad AX, immediate8`	`d5 Ib`	2	5

AAM

Flags updated

C	P	A	Z	S	T	I	D	O
	M		M	M				

Encoding

Table 18. AAM encoding

Mnemonic	Opcode	Size	Latency
`aam immediate8 (ex: 0x1)`	`d4 Ib`	2	18

AAS

Flags updated

C	P	A	Z	S	T	I	D	O
M		M						

Encoding

Table 19. AAS encoding

Mnemonic	Opcode	Size	Latency
`aas AX`	`3f`	1	2

ADC

Flags updated

C	P	A	Z	S	T	I	D	O
M	M	M	M	M				M

Encoding

Table 20. ADC encoding

Mnemonic	Opcode	Size	Latency
`adc register8, register8`	`10 /r`	2-4	2
`adc memory8, register8`	`10 /r`	2-4	8
`adc register16, register16`	`11 /r`	2-4	2
`adc memory16, register16`	`11 /r`	2-4	8
`adc register8, register8`	`12 /r`	2-4	2
`adc register8, memory8`	`12 /r`	2-4	5
`adc register16, register16`	`13 /r`	2-4	2
`adc register16, memory16`	`13 /r`	2-4	5
`adc AL, immediate8`	`14 Ib`	2	2
`adc AX, immediate16`	`15 Iw`	3	2
`adc register8, immediate8`	`80 /2 Ib`	3-5	3
`adc memory8, immediate8`	`80 /2 Ib`	3-5	9
`adc register16, immediate16`	`81 /2 Iw`	4-6	3
`adc memory16, immediate16`	`81 /2 Iw`	4-6	9
`adc register8, immediate8`	`82 /2 Ib`	3-5	3
`adc memory8, immediate8`	`82 /2 Ib`	3-5	9
`adc register16, immediate8`	`83 /2 Ib`	3-5	4
`adc memory16, immediate8`	`83 /2 Ib`	3-5	11

ADD

Flags updated

C	P	A	Z	S	T	I	D	O
M	M	M	M	M				M

Encoding

Table 21. ADD encoding

Mnemonic	Opcode	Size	Latency
`add register8, register8`	`00 /r`	2-4	2
`add memory8, register8`	`00 /r`	2-4	8
`add register16, register16`	`01 /r`	2-4	2
`add memory16, register16`	`01 /r`	2-4	8
`add register8, register8`	`02 /r`	2-4	2
`add register8, memory8`	`02 /r`	2-4	5
`add register16, register16`	`03 /r`	2-4	2
`add register16, memory16`	`03 /r`	2-4	5
`add AL, immediate8`	`04 Ib`	2	2
`add AX, immediate16`	`05 Iw`	3	2
`add register8, immediate8`	`80 /0 Ib`	3-5	3
`add memory8, immediate8`	`80 /0 Ib`	3-5	9
`add register16, immediate16`	`81 /0 Iw`	4-6	3
`add memory16, immediate16`	`81 /0 Iw`	4-6	9
`add register8, immediate8`	`82 /0 Ib`	3-5	3
`add memory8, immediate8`	`82 /0 Ib`	3-5	9
`add register16, immediate8`	`83 /0 Ib`	3-5	4
`add memory16, immediate8`	`83 /0 Ib`	3-5	11

AND

Flags updated

C	P	A	Z	S	T	I	D	O
M	M		M	M				M

Encoding

Table 22. AND encoding

Mnemonic	Opcode	Size	Latency
`and register8, register8`	`20 /r`	2-4	2
`and memory8, register8`	`20 /r`	2-4	8
`and register16, register16`	`21 /r`	2-4	2
`and memory16, register16`	`21 /r`	2-4	8
`and register8, register8`	`22 /r`	2-4	2
`and register8, memory8`	`22 /r`	2-4	5
`and register16, register16`	`23 /r`	2-4	2
`and register16, memory16`	`23 /r`	2-4	5
`and AL, immediate8`	`24 Ib`	2	2
`and AX, immediate16`	`25 Iw`	3	2
`and register8, immediate8`	`80 /4 Ib`	3-5	3
`and memory8, immediate8`	`80 /4 Ib`	3-5	9
`and register16, immediate16`	`81 /4 Iw`	4-6	3
`and memory16, immediate16`	`81 /4 Iw`	4-6	9
`and register8, immediate8`	`82 /4 Ib`	3-5	3
`and memory8, immediate8`	`82 /4 Ib`	3-5	9
`and register16, immediate8`	`83 /4 Ib`	3-5	4
`and memory16, immediate8`	`83 /4 Ib`	3-5	11

BOUND

Flags updated

None

Encoding

Table 23. BOUND encoding

Mnemonic	Opcode	Size	Latency
`bound memory(seg:offset)`	`62`	2	41

CALL

Flags updated

None

Encoding

Table 24. CALL encoding

Mnemonic	Opcode	Size	Latency
`call immediate16`	`e8 Iw`	3	6
`call displacement(seg:offset)`	`9a`	5	12
`call register16`	`ff /2`	2-4	8
`call memory16`	`ff /2`	2-4	12
`call memory(seg:offset)`	`ff /3`	2	22

CBW

Flags updated

None

Encoding

Table 25. CBW encoding

Mnemonic	Opcode	Size	Latency
`cbw AL`	`98`	1	2

CLC

Flags updated

C	P	A	Z	S	T	I	D	O
0								

Encoding

Table 26. CLC encoding

Mnemonic	Opcode	Size	Latency
`clc`	`f8`	1	1

CLD

Flags updated

C	P	A	Z	S	T	I	D	O
							0	

Encoding

Table 27. CLD encoding

Mnemonic	Opcode	Size	Latency
`cld`	`fc`	1	1

CLI

Flags updated

C	P	A	Z	S	T	I	D	O
						0		

Encoding

Table 28. CLI encoding

Mnemonic	Opcode	Size	Latency
`cli`	`fa`	1	1

CMC

Flags updated

C	P	A	Z	S	T	I	D	O
M								

Encoding

Table 29. CMC encoding

Mnemonic	Opcode	Size	Latency
`cmc`	`f5`	1	1

CMP

Flags updated

C	P	A	Z	S	T	I	D	O
M	M	M	M	M				M

Encoding

Table 30. CMP encoding

Mnemonic	Opcode	Size	Latency
`cmp register8, register8`	`38 /r`	2-4	2
`cmp memory8, register8`	`38 /r`	2-4	5
`cmp register16, register16`	`39 /r`	2-4	2
`cmp memory16, register16`	`39 /r`	2-4	5
`cmp register8, register8`	`3a /r`	2-4	2
`cmp register8, memory8`	`3a /r`	2-4	5
`cmp register16, register16`	`3b /r`	2-4	2
`cmp register16, memory16`	`3b /r`	2-4	5
`cmp AL, immediate8`	`3c Ib`	2	2
`cmp AX, immediate16`	`3d Iw`	3	2
`cmp register8, immediate8`	`80 /7 Ib`	3-5	3
`cmp memory8, immediate8`	`80 /7 Ib`	3-5	6
`cmp register16, immediate16`	`81 /7 Iw`	4-6	3
`cmp memory16, immediate16`	`81 /7 Iw`	4-6	6
`cmp register8, immediate8`	`82 /7 Ib`	3-5	3
`cmp memory8, immediate8`	`82 /7 Ib`	3-5	6
`cmp register16, immediate8`	`83 /7 Ib`	3-5	4
`cmp memory16, immediate8`	`83 /7 Ib`	3-5	8

CMPS

Flags updated

C	P	A	Z	S	T	I	D	O
M	M	M	M	M				M

Encoding

Table 31. CMPS encoding

Mnemonic	Opcode	Size	Latency
`cmps`	`a6`	1	18 + (n-1)*14
`cmps`	`a7`	1	17 + (n-1)*13

CWD

Flags updated

None

Encoding

Table 32. CWD encoding

Mnemonic	Opcode	Size	Latency
`cwd AX`	`99`	1	2

DAA

Flags updated

C	P	A	Z	S	T	I	D	O
M	M	M	M	M				

Encoding

Table 33. DAA encoding

Mnemonic	Opcode	Size	Latency
`daa AL`	`27`	1	2

DAS

Flags updated

C	P	A	Z	S	T	I	D	O
M	M	M	M	M				

Encoding

Table 34. DAS encoding

Mnemonic	Opcode	Size	Latency
`das AX`	`2f`	1	2

DEC

Flags updated

C	P	A	Z	S	T	I	D	O
	M	M	M	M				M

Encoding

Table 35. DEC encoding

Mnemonic	Opcode	Size	Latency
`dec register8`	`fe /1`	2-4	3
`dec memory8`	`fe /1`	2-4	9
`dec register16`	`ff /1`	2-4	3
`dec memory16`	`ff /1`	2-4	13
`dec AX`	`48`	1	2
`dec CX`	`49`	1	2
`dec DX`	`4a`	1	2
`dec BX`	`4b`	1	2
`dec SP`	`4c`	1	2
`dec BP`	`4d`	1	2
`dec SI`	`4e`	1	2
`dec DI`	`4f`	1	2

DIV

Flags updated

None

Encoding

Table 36. DIV encoding

Mnemonic	Opcode	Size	Latency
`div register8 (ex: 0xa)`	`f6 /6`	2-4	18
`div memory8 (ex: 0xa)`	`f6 /6`	2-4	20
`div register16 (ex: 0xa)`	`f7 /6`	2-4	25
`div memory16 (ex: 0xa)`	`f7 /6`	2-4	27

ENTER

Flags updated

None

Encoding

Table 37. ENTER encoding

Mnemonic	Opcode	Size	Latency
`enter immediate16, immediate8 (ex: 0x0 0x0)`	`c8 Iw, Ib`	4	9
`enter immediate16, immediate8 (ex: 0x20 0x0)`	`c8 Iw, Ib`	4	9
`enter immediate16, immediate8 (ex: 0x0 0x1)`	`c8 Iw, Ib`	4	20
`enter immediate16, immediate8 (ex: 0x0 0x2)`	`c8 Iw, Ib`	4	30

ESC

Flags updated

None

Encoding

Table 38. ESC encoding

Mnemonic	Opcode	Size	Latency
`esc register8`	`d8 /r`	2-4	2
`esc memory8`	`d8 /r`	2-4	2
`esc register16`	`d9 /r`	2-4	2
`esc memory16`	`d9 /r`	2-4	2
`esc register8`	`da /r`	2-4	2
`esc memory8`	`da /r`	2-4	2
`esc register16`	`db /r`	2-4	2
`esc memory16`	`db /r`	2-4	2
`esc register8`	`dc /r`	2-4	2
`esc memory8`	`dc /r`	2-4	2
`esc register16`	`dd /r`	2-4	2
`esc memory16`	`dd /r`	2-4	2
`esc register8`	`de /r`	2-4	2
`esc memory8`	`de /r`	2-4	2
`esc register16`	`df /r`	2-4	2
`esc memory16`	`df /r`	2-4	2

NOTE Acts as a NOP unless built with -S80X86_TRAP_ESCAPE=ON otherwise raises trap 7.

HLT

Flags updated

None

Encoding

Table 39. HLT encoding

Mnemonic	Opcode	Size	Latency
`hlt`	`f4`	1	-

IDIV

Flags updated

None

Encoding

Table 40. IDIV encoding

Mnemonic	Opcode	Size	Latency
`idiv register8 (ex: 0xa)`	`f6 /7`	2-4	19
`idiv memory8 (ex: 0xa)`	`f6 /7`	2-4	21
`idiv register16 (ex: 0xa)`	`f7 /7`	2-4	26
`idiv memory16 (ex: 0xa)`	`f7 /7`	2-4	28

IMUL

Flags updated

C	P	A	Z	S	T	I	D	O
M			M					M

Encoding

Table 41. IMUL encoding

Mnemonic	Opcode	Size	Latency
`imul register8`	`f6 /5`	2-4	6
`imul memory8`	`f6 /5`	2-4	8
`imul register16`	`f7 /5`	2-4	6
`imul memory16`	`f7 /5`	2-4	8
`imul register16, immediate16`	`69 /r Iw`	4-6	2
`imul memory16, immediate16`	`69 /r Iw`	4-6	5
`imul register16, immediate8`	`6b /r Ib`	3-5	2
`imul memory16, immediate8`	`6b /r Ib`	3-5	5

INB

Flags updated

None

Encoding

Table 42. INB encoding

Mnemonic	Opcode	Size	Latency
`inb immediate8`	`e4 Ib`	2	6
`inb DX`	`ec`	1	6

INC

Flags updated

C	P	A	Z	S	T	I	D	O
	M	M	M	M				M

Encoding

Table 43. INC encoding

Mnemonic	Opcode	Size	Latency
`inc register8`	`fe /0`	2-4	3
`inc memory8`	`fe /0`	2-4	9
`inc register16`	`ff /0`	2-4	3
`inc memory16`	`ff /0`	2-4	13
`inc AX`	`40`	1	2
`inc CX`	`41`	1	2
`inc DX`	`42`	1	2
`inc BX`	`43`	1	2
`inc SP`	`44`	1	2
`inc BP`	`45`	1	2
`inc SI`	`46`	1	2
`inc DI`	`47`	1	2

INS

Flags updated

None

Encoding

Table 44. INS encoding

Mnemonic	Opcode	Size	Latency
`ins`	`6c`	1	19 + (n-1)*14
`ins`	`6d`	1	19 + (n-1)*14

INT

Flags updated

None

Encoding

Table 45. INT encoding

Mnemonic	Opcode	Size	Latency
`int immediate8`	`cd Ib`	2	31

INT3

Flags updated

None

Encoding

Table 46. INT3 encoding

Mnemonic	Opcode	Size	Latency
`int3`	`cc`	1	29

INTO

Flags updated

None

Encoding

Table 47. INTO encoding

Mnemonic	Opcode	Size	Latency
`into`	`ce`	1	3

INW

Flags updated

None

Encoding

Table 48. INW encoding

Mnemonic	Opcode	Size	Latency
`inw immediate8`	`e5 Ib`	2	6
`inw DX`	`ed`	1	6

IRET

Flags updated

C	P	A	Z	S	T	I	D	O
M	M	M	M	M	M	M	M	M

Encoding

Table 49. IRET encoding

Mnemonic	Opcode	Size	Latency
`iret`	`cf`	1	15

JB

Flags updated

None

Encoding

Table 50. JB encoding

Mnemonic	Opcode	Size	Latency
`jb immediate8`	`72 Ib`	2	3 (taken) / 3 (not taken)

JBE

Flags updated

None

Encoding

Table 51. JBE encoding

Mnemonic	Opcode	Size	Latency
`jbe immediate8`	`76 Ib`	2	3 (taken) / 3 (not taken)

JCXZ

Flags updated

None

Encoding

Table 52. JCXZ encoding

Mnemonic	Opcode	Size	Latency
`jcxz immediate8`	`e3 Ib`	2	3 (taken) / 3 (not taken)

JE

Flags updated

None

Encoding

Table 53. JE encoding

Mnemonic	Opcode	Size	Latency
`je immediate8`	`74 Ib`	2	3 (taken) / 3 (not taken)

JL

Flags updated

None

Encoding

Table 54. JL encoding

Mnemonic	Opcode	Size	Latency
`jl immediate8`	`7c Ib`	2	3 (taken) / 3 (not taken)

JLE

Flags updated

None

Encoding

Table 55. JLE encoding

Mnemonic	Opcode	Size	Latency
`jle immediate8`	`7e Ib`	2	3 (taken) / 3 (not taken)

JMP

Flags updated

None

Encoding

Table 56. JMP encoding

Mnemonic	Opcode	Size	Latency
`jmp immediate16`	`e9 Iw`	3	1
`jmp immediate16, immediate16`	`ea Iw, Iw`	5	2
`jmp immediate8`	`eb Ib`	2	1
`jmp register16`	`ff /4`	2-4	3
`jmp memory16`	`ff /4`	2-4	6
`jmp memory(seg:offset)`	`ff /5`	2	11

JNB

Flags updated

None

Encoding

Table 57. JNB encoding

Mnemonic	Opcode	Size	Latency
`jnb immediate8`	`73 Ib`	2	3 (taken) / 3 (not taken)

JNBE

Flags updated

None

Encoding

Table 58. JNBE encoding

Mnemonic	Opcode	Size	Latency
`jnbe immediate8`	`77 Ib`	2	3 (taken) / 3 (not taken)

JNE

Flags updated

None

Encoding

Table 59. JNE encoding

Mnemonic	Opcode	Size	Latency
`jne immediate8`	`75 Ib`	2	3 (taken) / 3 (not taken)

JNL

Flags updated

None

Encoding

Table 60. JNL encoding

Mnemonic	Opcode	Size	Latency
`jnl immediate8`	`7d Ib`	2	3 (taken) / 3 (not taken)

JNLE

Flags updated

None

Encoding

Table 61. JNLE encoding

Mnemonic	Opcode	Size	Latency
`jnle immediate8`	`7f Ib`	2	3 (taken) / 3 (not taken)

JNO

Flags updated

None

Encoding

Table 62. JNO encoding

Mnemonic	Opcode	Size	Latency
`jno immediate8`	`71 Ib`	2	3 (taken) / 3 (not taken)

JNP

Flags updated

None

Encoding

Table 63. JNP encoding

Mnemonic	Opcode	Size	Latency
`jnp immediate8`	`7b Ib`	2	3 (taken) / 3 (not taken)

JNS

Flags updated

None

Encoding

Table 64. JNS encoding

Mnemonic	Opcode	Size	Latency
`jns immediate8`	`79 Ib`	2	3 (taken) / 3 (not taken)

JO

Flags updated

None

Encoding

Table 65. JO encoding

Mnemonic	Opcode	Size	Latency
`jo immediate8`	`70 Ib`	2	3 (taken) / 3 (not taken)

JP

Flags updated

None

Encoding

Table 66. JP encoding

Mnemonic	Opcode	Size	Latency
`jp immediate8`	`7a Ib`	2	3 (taken) / 3 (not taken)

JS

Flags updated

None

Encoding

Table 67. JS encoding

Mnemonic	Opcode	Size	Latency
`js immediate8`	`78 Ib`	2	3 (taken) / 3 (not taken)

LAHF

Flags updated

None

Encoding

Table 68. LAHF encoding

Mnemonic	Opcode	Size	Latency
`lahf AH`	`9f`	1	1

LDS

Flags updated

None

Encoding

Table 69. LDS encoding

Mnemonic	Opcode	Size	Latency
`lds memory16, memory(seg:offset)`	`c5 /r`	2-4	10

LEA

Flags updated

None

Encoding

Table 70. LEA encoding

Mnemonic	Opcode	Size	Latency
`lea memory16, memory16`	`8d /r`	2-4	2

LEAVE

Flags updated

None

Encoding

Table 71. LEAVE encoding

Mnemonic	Opcode	Size	Latency
`leave`	`c9`	1	7

LES

Flags updated

None

Encoding

Table 72. LES encoding

Mnemonic	Opcode	Size	Latency
`les memory16, memory(seg:offset)`	`c4 /r`	2-4	10

LODS

Flags updated

None

Encoding

Table 73. LODS encoding

Mnemonic	Opcode	Size	Latency
`lods`	`ac`	1	13 + (n-1)*8
`lods`	`ad`	1	13 + (n-1)*8

LOOP

Flags updated

None

Encoding

Table 74. LOOP encoding

Mnemonic	Opcode	Size	Latency
`loop immediate8`	`e2 Ib`	2	4 (taken) / 4 (not taken)

LOOPE

Flags updated

None

Encoding

Table 75. LOOPE encoding

Mnemonic	Opcode	Size	Latency
`loope immediate8`	`e1 Ib`	2	4 (taken) / 4 (not taken)

LOOPNE

Flags updated

None

Encoding

Table 76. LOOPNE encoding

Mnemonic	Opcode	Size	Latency
`loopne immediate8`	`e0 Ib`	2	4 (taken) / 3 (not taken)

MOV

Flags updated

None

Encoding

Table 77. MOV encoding

Mnemonic	Opcode	Size	Latency
`mov register8, register8`	`88 /r`	2-4	2
`mov memory8, register8`	`88 /r`	2-4	5
`mov register16, register16`	`89 /r`	2-4	2
`mov memory16, register16`	`89 /r`	2-4	5
`mov register8, register8`	`8a /r`	2-4	2
`mov register8, memory8`	`8a /r`	2-4	5
`mov register16, register16`	`8b /r`	2-4	2
`mov register16, memory16`	`8b /r`	2-4	5
`mov register8, immediate8`	`c6 /0 Ib`	3-5	3
`mov memory8, immediate8`	`c6 /0 Ib`	3-5	7
`mov register16, immediate16`	`c7 /0 Iw`	4-6	3
`mov memory16, immediate16`	`c7 /0 Iw`	4-6	7
`mov AL, immediate8`	`b0 Ib`	2	1
`mov CL, immediate8`	`b1 Ib`	2	1
`mov DL, immediate8`	`b2 Ib`	2	1
`mov BL, immediate8`	`b3 Ib`	2	1
`mov AH, immediate8`	`b4 Ib`	2	1
`mov CH, immediate8`	`b5 Ib`	2	1
`mov DH, immediate8`	`b6 Ib`	2	1
`mov BH, immediate8`	`b7 Ib`	2	1
`mov AX, immediate16`	`b8 Iw`	3	1
`mov CX, immediate16`	`b9 Iw`	3	1
`mov DX, immediate16`	`ba Iw`	3	1
`mov BX, immediate16`	`bb Iw`	3	1
`mov SP, immediate16`	`bc Iw`	3	1
`mov BP, immediate16`	`bd Iw`	3	1
`mov SI, immediate16`	`be Iw`	3	1
`mov DI, immediate16`	`bf Iw`	3	1

Mnemonic	Opcode	Size	Latency
`mov ES, register16`	`8e /0`	2-4	2
`mov ES, memory16`	`8e /0`	2-4	5
`mov CS, register16`	`8e /1`	2-4	2
`mov CS, memory16`	`8e /1`	2-4	5
`mov SS, register16`	`8e /2`	2-4	2
`mov SS, memory16`	`8e /2`	2-4	5
`mov DS, register16`	`8e /3`	2-4	2
`mov DS, memory16`	`8e /3`	2-4	5
`mov ES, register16`	`8e /4`	2-4	2
`mov ES, memory16`	`8e /4`	2-4	5
`mov CS, register16`	`8e /5`	2-4	2
`mov CS, memory16`	`8e /5`	2-4	5
`mov SS, register16`	`8e /6`	2-4	2
`mov SS, memory16`	`8e /6`	2-4	5
`mov DS, register16`	`8e /7`	2-4	2
`mov DS, memory16`	`8e /7`	2-4	5
`mov register16, ES`	`8c /0`	2-4	3
`mov memory16, ES`	`8c /0`	2-4	7
`mov register16, CS`	`8c /1`	2-4	3
`mov memory16, CS`	`8c /1`	2-4	9
`mov register16, SS`	`8c /2`	2-4	3
`mov memory16, SS`	`8c /2`	2-4	7
`mov register16, DS`	`8c /3`	2-4	3
`mov memory16, DS`	`8c /3`	2-4	7
`mov register16, ES`	`8c /4`	2-4	3
`mov memory16, ES`	`8c /4`	2-4	7
`mov register16, CS`	`8c /5`	2-4	3
`mov memory16, CS`	`8c /5`	2-4	9
`mov register16, SS`	`8c /6`	2-4	3
`mov memory16, SS`	`8c /6`	2-4	7
`mov register16, DS`	`8c /7`	2-4	3
`mov memory16, DS`	`8c /7`	2-4	7
`mov AL, moffset8`	`a0`	1	5
`mov AX, moffset16`	`a1`	1	5

Mnemonic	Opcode	Size	Latency
`mov moffset16, AL`	`a2`	1	5
`mov moffset16, AX`	`a3`	1	5

MOVS

Flags updated

None

Encoding

Table 78. MOVS encoding

Mnemonic	Opcode	Size	Latency
`movs`	`a4`	1	16 + (n-1)*12
`movs`	`a5`	1	17 + (n-1)*12

MUL

Flags updated

C	P	A	Z	S	T	I	D	O
M			M					M

Encoding

Table 79. MUL encoding

Mnemonic	Opcode	Size	Latency
`mul register8`	`f6 /4`	2-4	6
`mul memory8`	`f6 /4`	2-4	8
`mul register16`	`f7 /4`	2-4	6
`mul memory16`	`f7 /4`	2-4	8

NEG

Flags updated

C	P	A	Z	S	T	I	D	O
M	M	M	M	M				M

Encoding

Table 80. NEG encoding

Mnemonic	Opcode	Size	Latency
`neg register8`	`f6 /3`	2-4	3
`neg memory8`	`f6 /3`	2-4	9
`neg register16`	`f7 /3`	2-4	3
`neg memory16`	`f7 /3`	2-4	9

NOP

Flags updated

None

Encoding

Table 81. NOP encoding

Mnemonic	Opcode	Size	Latency
`nop`	`90`	1	1

NOT

Flags updated

C	P	A	Z	S	T	I	D	O
M	M	M	M	M				M

Encoding

Table 82. NOT encoding

Mnemonic	Opcode	Size	Latency
`not register8`	`f6 /2`	2-4	3
`not memory8`	`f6 /2`	2-4	9
`not register16`	`f7 /2`	2-4	3
`not memory16`	`f7 /2`	2-4	13

OR

Flags updated

C	P	A	Z	S	T	I	D	O
M	M		M	M				M

Encoding

Table 83. OR encoding

Mnemonic	Opcode	Size	Latency
`or register8, register8`	`08 /r`	2-4	2
`or memory8, register8`	`08 /r`	2-4	8
`or register16, register16`	`09 /r`	2-4	2
`or memory16, register16`	`09 /r`	2-4	8
`or register8, register8`	`0a /r`	2-4	2
`or register8, memory8`	`0a /r`	2-4	5
`or register16, register16`	`0b /r`	2-4	2
`or register16, memory16`	`0b /r`	2-4	5
`or AL, immediate8`	`0c Ib`	2	2
`or AX, immediate16`	`0d Iw`	3	2
`or register8, immediate8`	`80 /1 Ib`	3-5	3
`or memory8, immediate8`	`80 /1 Ib`	3-5	9
`or register16, immediate16`	`81 /1 Iw`	4-6	3
`or memory16, immediate16`	`81 /1 Iw`	4-6	9
`or register8, immediate8`	`82 /1 Ib`	3-5	3
`or memory8, immediate8`	`82 /1 Ib`	3-5	9
`or register16, immediate8`	`83 /1 Ib`	3-5	4
`or memory16, immediate8`	`83 /1 Ib`	3-5	11

OUTB

Flags updated

None

Encoding

Table 84. OUTB encoding

Mnemonic	Opcode	Size	Latency
`outb immediate8`	`e6 Ib`	2	6
`outb DX`	`ee`	1	6

OUTS

Flags updated

None

Encoding

Table 85. OUTS encoding

Mnemonic	Opcode	Size	Latency
`outs`	`6e`	1	18 + (n-1)*13
`outs`	`6f`	1	18 + (n-1)*13

OUTW

Flags updated

None

Encoding

Table 86. OUTW encoding

Mnemonic	Opcode	Size	Latency
`outw immediate8`	`e7 Ib`	2	6
`outw DX`	`ef`	1	6

POP

Flags updated

None

Encoding

Table 87. POP encoding

Mnemonic	Opcode	Size	Latency
`pop AX`	`58`	1	6
`pop CX`	`59`	1	6
`pop DX`	`5a`	1	6
`pop BX`	`5b`	1	6
`pop SP`	`5c`	1	6
`pop BP`	`5d`	1	6
`pop SI`	`5e`	1	6
`pop DI`	`5f`	1	6
`pop ES`	`07`	1	6
`pop SS`	`17`	1	6
`pop DS`	`1f`	1	6
`pop register16`	`8f /r`	2-4	9
`pop memory16`	`8f /r`	2-4	12

POPA

Flags updated

None

Encoding

Table 88. POPA encoding

Mnemonic	Opcode	Size	Latency
`popa`	`61`	1	30

POPF

Flags updated

C	P	A	Z	S	T	I	D	O
M	M	M	M	M	M	M	M	M

Encoding

Table 89. POPF encoding

Mnemonic	Opcode	Size	Latency
`popf`	`9d`	1	6

PUSH

Flags updated

None

Encoding

Table 90. PUSH encoding

Mnemonic	Opcode	Size	Latency
`push register16`	`ff /6`	2-4	7
`push memory16`	`ff /6`	2-4	9
`push ES`	`06`	1	6
`push CS`	`0e`	1	6
`push SS`	`16`	1	6
`push DS`	`1e`	1	6
`push AX`	`50`	1	6
`push CX`	`51`	1	6
`push DX`	`52`	1	6
`push BX`	`53`	1	6
`push SP`	`54`	1	5
`push BP`	`55`	1	6
`push SI`	`56`	1	6
`push DI`	`57`	1	6
`push immediate16`	`68 Iw`	3	6
`push immediate8`	`6a Ib`	2	7

NOTE The core implements the original 8086/80186 behaviour of pushing the decremented value of SP in a 'push sp' instruction rather than the original value.

PUSHA

Flags updated

None

Encoding

Table 91. PUSHA encoding

Mnemonic	Opcode	Size	Latency
`pusha`	`60`	1	34

PUSHF

Flags updated

None

Encoding

Table 92. PUSHF encoding

Mnemonic	Opcode	Size	Latency
`pushf`	`9c`	1	5

RCL

Flags updated

C	P	A	Z	S	T	I	D	O
M								M

Encoding

Table 93. RCL encoding

Mnemonic	Opcode	Size	Latency
`rcl register8`	`d0 /2`	2-4	3
`rcl memory8`	`d0 /2`	2-4	9
`rcl register16`	`d1 /2`	2-4	3
`rcl memory16`	`d1 /2`	2-4	9
`rcl register8, immediate8`	`c0 /2 Ib`	3-5	3 + (n-1)
`rcl memory8, immediate8`	`c0 /2 Ib`	3-5	9 + (n-1)
`rcl register16, immediate8`	`c1 /2 Ib`	3-5	4 + (n-1)
`rcl memory16, immediate8`	`c1 /2 Ib`	3-5	11 + (n-1)
`rcl register8, CL`	`d2 /2`	2-4	4 + (n-1)
`rcl memory8, CL`	`d2 /2`	2-4	11 + (n-1)
`rcl register16, CL`	`d3 /2`	2-4	4 + (n-1)
`rcl memory16, CL`	`d3 /2`	2-4	11 + (n-1)

RCR

Flags updated

C	P	A	Z	S	T	I	D	O
M								M

Encoding

Table 94. RCR encoding

Mnemonic	Opcode	Size	Latency
`rcr register8`	`d0 /3`	2-4	3
`rcr memory8`	`d0 /3`	2-4	9
`rcr register16`	`d1 /3`	2-4	3
`rcr memory16`	`d1 /3`	2-4	9
`rcr register8, immediate8`	`c0 /3 Ib`	3-5	3 + (n-1)
`rcr memory8, immediate8`	`c0 /3 Ib`	3-5	9 + (n-1)
`rcr register16, immediate8`	`c1 /3 Ib`	3-5	4 + (n-1)
`rcr memory16, immediate8`	`c1 /3 Ib`	3-5	11 + (n-1)
`rcr register8, CL`	`d2 /3`	2-4	4 + (n-1)
`rcr memory8, CL`	`d2 /3`	2-4	11 + (n-1)
`rcr register16, CL`	`d3 /3`	2-4	4 + (n-1)
`rcr memory16, CL`	`d3 /3`	2-4	11 + (n-1)

RET

Flags updated

None

Encoding

Table 95. RET encoding

Mnemonic	Opcode	Size	Latency
`ret`	`c3`	1	6
`ret immediate16`	`c2 Iw`	3	7
`ret`	`cb`	1	10
`ret immediate16`	`ca Iw`	3	11

ROL

Flags updated

C	P	A	Z	S	T	I	D	O
M								M

Encoding

Table 96. ROL encoding

Mnemonic	Opcode	Size	Latency
`rol register8`	`d0 /0`	2-4	3
`rol memory8`	`d0 /0`	2-4	9
`rol register16`	`d1 /0`	2-4	3
`rol memory16`	`d1 /0`	2-4	9
`rol register8, immediate8`	`c0 /0 Ib`	3-5	3 + (n-1)
`rol memory8, immediate8`	`c0 /0 Ib`	3-5	9 + (n-1)
`rol register16, immediate8`	`c1 /0 Ib`	3-5	4 + (n-1)
`rol memory16, immediate8`	`c1 /0 Ib`	3-5	11 + (n-1)
`rol register8, CL`	`d2 /0`	2-4	4 + (n-1)
`rol memory8, CL`	`d2 /0`	2-4	11 + (n-1)
`rol register16, CL`	`d3 /0`	2-4	4 + (n-1)
`rol memory16, CL`	`d3 /0`	2-4	11 + (n-1)

ROR

Flags updated

C	P	A	Z	S	T	I	D	O
M								M

Encoding

Table 97. ROR encoding

Mnemonic	Opcode	Size	Latency
`ror register8`	`d0 /1`	2-4	3
`ror memory8`	`d0 /1`	2-4	9
`ror register16`	`d1 /1`	2-4	3
`ror memory16`	`d1 /1`	2-4	9
`ror register8, immediate8`	`c0 /1 Ib`	3-5	3 + (n-1)
`ror memory8, immediate8`	`c0 /1 Ib`	3-5	9 + (n-1)
`ror register16, immediate8`	`c1 /1 Ib`	3-5	4 + (n-1)
`ror memory16, immediate8`	`c1 /1 Ib`	3-5	11 + (n-1)
`ror register8, CL`	`d2 /1`	2-4	4 + (n-1)
`ror memory8, CL`	`d2 /1`	2-4	11 + (n-1)
`ror register16, CL`	`d3 /1`	2-4	4 + (n-1)
`ror memory16, CL`	`d3 /1`	2-4	11 + (n-1)

SAHF

Flags updated

C	P	A	Z	S	T	I	D	O
M	M	M	M	M				

Encoding

Table 98. SAHF encoding

Mnemonic	Opcode	Size	Latency
`sahf AH`	`9e`	1	2

SAL

Flags updated

C	P	A	Z	S	T	I	D	O
M	M		M	M				M

Encoding

Table 99. SAL encoding

Mnemonic	Opcode	Size	Latency
`sal register8`	`d0 /6`	2-4	3
`sal memory8`	`d0 /6`	2-4	9
`sal register16`	`d1 /6`	2-4	3
`sal memory16`	`d1 /6`	2-4	9
`sal register8, immediate8`	`c0 /6 Ib`	3-5	3 + (n-1)
`sal memory8, immediate8`	`c0 /6 Ib`	3-5	9 + (n-1)
`sal register16, immediate8`	`c1 /6 Ib`	3-5	4 + (n-1)
`sal memory16, immediate8`	`c1 /6 Ib`	3-5	11 + (n-1)
`sal register8, CL`	`d2 /6`	2-4	4 + (n-1)
`sal memory8, CL`	`d2 /6`	2-4	11 + (n-1)
`sal register16, CL`	`d3 /6`	2-4	4 + (n-1)
`sal memory16, CL`	`d3 /6`	2-4	11 + (n-1)

SAR

Flags updated

C	P	A	Z	S	T	I	D	O
M	M		M	M				M

Encoding

Table 100. SAR encoding

Mnemonic	Opcode	Size	Latency
`sar register8`	`d0 /7`	2-4	3
`sar memory8`	`d0 /7`	2-4	9
`sar register16`	`d1 /7`	2-4	3
`sar memory16`	`d1 /7`	2-4	9
`sar register8, immediate8`	`c0 /7 Ib`	3-5	3 + (n-1)
`sar memory8, immediate8`	`c0 /7 Ib`	3-5	9 + (n-1)
`sar register16, immediate8`	`c1 /7 Ib`	3-5	4 + (n-1)
`sar memory16, immediate8`	`c1 /7 Ib`	3-5	11 + (n-1)
`sar register8, CL`	`d2 /7`	2-4	4 + (n-1)
`sar memory8, CL`	`d2 /7`	2-4	11 + (n-1)
`sar register16, CL`	`d3 /7`	2-4	4 + (n-1)
`sar memory16, CL`	`d3 /7`	2-4	11 + (n-1)

SBB

Flags updated

C	P	A	Z	S	T	I	D	O
M	M	M	M	M				M

Encoding

Table 101. SBB encoding

Mnemonic	Opcode	Size	Latency
`sbb register8, register8`	`18 /r`	2-4	2
`sbb memory8, register8`	`18 /r`	2-4	8
`sbb register16, register16`	`19 /r`	2-4	2
`sbb memory16, register16`	`19 /r`	2-4	8
`sbb register8, register8`	`1a /r`	2-4	2
`sbb register8, memory8`	`1a /r`	2-4	5
`sbb register16, register16`	`1b /r`	2-4	2
`sbb register16, memory16`	`1b /r`	2-4	5
`sbb AL, immediate8`	`1c Ib`	2	2
`sbb AX, immediate16`	`1d Iw`	3	2
`sbb register8, immediate8`	`80 /3 Ib`	3-5	3
`sbb memory8, immediate8`	`80 /3 Ib`	3-5	9
`sbb register16, immediate16`	`81 /3 Iw`	4-6	3
`sbb memory16, immediate16`	`81 /3 Iw`	4-6	9
`sbb register8, immediate8`	`82 /3 Ib`	3-5	3
`sbb memory8, immediate8`	`82 /3 Ib`	3-5	9
`sbb register16, immediate8`	`83 /3 Ib`	3-5	4
`sbb memory16, immediate8`	`83 /3 Ib`	3-5	11

SCAS

Flags updated

C	P	A	Z	S	T	I	D	O
M	M	M	M	M				M

Encoding

Table 102. SCAS encoding

Mnemonic	Opcode	Size	Latency
`scas`	`ae`	1	11 + (n-1)*7
`scas`	`af`	1	11 + (n-1)*7

SETALC

Flags updated

C	P	A	Z	S	T	I	D	O
M								

Encoding

Table 103. SETALC encoding

Mnemonic	Opcode	Size	Latency
`setalc AL`	`d6`	1	2

SHL

Flags updated

C	P	A	Z	S	T	I	D	O
M	M		M	M				M

Encoding

Table 104. SHL encoding

Mnemonic	Opcode	Size	Latency
`shl register8`	`d0 /4`	2-4	3
`shl memory8`	`d0 /4`	2-4	9
`shl register16`	`d1 /4`	2-4	3
`shl memory16`	`d1 /4`	2-4	9
`shl register8, immediate8`	`c0 /4 Ib`	3-5	3 + (n-1)
`shl memory8, immediate8`	`c0 /4 Ib`	3-5	9 + (n-1)
`shl register16, immediate8`	`c1 /4 Ib`	3-5	4 + (n-1)
`shl memory16, immediate8`	`c1 /4 Ib`	3-5	11 + (n-1)
`shl register8, CL`	`d2 /4`	2-4	4 + (n-1)
`shl memory8, CL`	`d2 /4`	2-4	11 + (n-1)
`shl register16, CL`	`d3 /4`	2-4	4 + (n-1)
`shl memory16, CL`	`d3 /4`	2-4	11 + (n-1)

SHR

Flags updated

C	P	A	Z	S	T	I	D	O
M	M		M	M				M

Encoding

Table 105. SHR encoding

Mnemonic	Opcode	Size	Latency
`shr register8`	`d0 /5`	2-4	3
`shr memory8`	`d0 /5`	2-4	9
`shr register16`	`d1 /5`	2-4	3
`shr memory16`	`d1 /5`	2-4	9
`shr register8, immediate8`	`c0 /5 Ib`	3-5	3 + (n-1)
`shr memory8, immediate8`	`c0 /5 Ib`	3-5	9 + (n-1)
`shr register16, immediate8`	`c1 /5 Ib`	3-5	4 + (n-1)
`shr memory16, immediate8`	`c1 /5 Ib`	3-5	11 + (n-1)
`shr register8, CL`	`d2 /5`	2-4	4 + (n-1)
`shr memory8, CL`	`d2 /5`	2-4	11 + (n-1)
`shr register16, CL`	`d3 /5`	2-4	4 + (n-1)
`shr memory16, CL`	`d3 /5`	2-4	11 + (n-1)

STC

Flags updated

C	P	A	Z	S	T	I	D	O
1								

Encoding

Table 106. STC encoding

Mnemonic	Opcode	Size	Latency
`stc`	`f9`	1	1

STD

Flags updated

C	P	A	Z	S	T	I	D	O
							1	

Encoding

Table 107. STD encoding

Mnemonic	Opcode	Size	Latency
`std`	`fd`	1	1

STI

Flags updated

C	P	A	Z	S	T	I	D	O
						1		

Encoding

Table 108. STI encoding

Mnemonic	Opcode	Size	Latency
`sti`	`fb`	1	1

STOS

Flags updated

None

Encoding

Table 109. STOS encoding

Mnemonic	Opcode	Size	Latency
`stos AL`	`aa`	1	12
`stos AX`	`ab`	1	9

SUB

Flags updated

C	P	A	Z	S	T	I	D	O
M	M	M	M	M				M

Encoding

Table 110. SUB encoding

Mnemonic	Opcode	Size	Latency
`sub register8, register8`	`28 /r`	2-4	2
`sub memory8, register8`	`28 /r`	2-4	8
`sub register16, register16`	`29 /r`	2-4	2
`sub memory16, register16`	`29 /r`	2-4	8
`sub register8, register8`	`2a /r`	2-4	2
`sub register8, memory8`	`2a /r`	2-4	5
`sub register16, register16`	`2b /r`	2-4	2
`sub register16, memory16`	`2b /r`	2-4	5
`sub AL, immediate8`	`2c Ib`	2	2
`sub AX, immediate16`	`2d Iw`	3	2
`sub register8, immediate8`	`80 /5 Ib`	3-5	3
`sub memory8, immediate8`	`80 /5 Ib`	3-5	9
`sub register16, immediate16`	`81 /5 Iw`	4-6	3
`sub memory16, immediate16`	`81 /5 Iw`	4-6	9
`sub register8, immediate8`	`82 /5 Ib`	3-5	3
`sub memory8, immediate8`	`82 /5 Ib`	3-5	9
`sub register16, immediate8`	`83 /5 Ib`	3-5	4
`sub memory16, immediate8`	`83 /5 Ib`	3-5	11

TEST

Flags updated

C	P	A	Z	S	T	I	D	O
M	M		M	M				M

Encoding

Table 111. TEST encoding

Mnemonic	Opcode	Size	Latency
`test register8, register8`	`84 /r`	2-4	2
`test register8, memory8`	`84 /r`	2-4	5
`test register16, register16`	`85 /r`	2-4	2
`test register16, memory16`	`85 /r`	2-4	5
`test AL, immediate8`	`a8 Ib`	2	2
`test AX, immediate16`	`a9 Iw`	3	2
`test register8, immediate8`	`f6 /0 Ib`	3-5	3
`test memory8, immediate8`	`f6 /0 Ib`	3-5	6
`test register8, immediate8`	`f6 /1 Ib`	3-5	3
`test memory8, immediate8`	`f6 /1 Ib`	3-5	6
`test register16, immediate16`	`f7 /0 Iw`	4-6	3
`test memory16, immediate16`	`f7 /0 Iw`	4-6	6
`test register16, immediate16`	`f7 /1 Iw`	4-6	3
`test memory16, immediate16`	`f7 /1 Iw`	4-6	6

WAIT

Flags updated

None

Encoding

Table 112. WAIT encoding

Mnemonic	Opcode	Size	Latency
`wait`	`9b`	1	-

XCHG

Flags updated

None

Encoding

Table 113. XCHG encoding

Mnemonic	Opcode	Size	Latency
`xchg register8, register8`	`86 /r`	2-4	4
`xchg register8, memory8`	`86 /r`	2-4	11
`xchg register16, register16`	`87 /r`	2-4	4
`xchg register16, memory16`	`87 /r`	2-4	11
`xchg CX, AX`	`91`	1	4
`xchg DX, AX`	`92`	1	4
`xchg BX, AX`	`93`	1	4
`xchg SP, AX`	`94`	1	4
`xchg BP, AX`	`95`	1	4
`xchg SI, AX`	`96`	1	4
`xchg DI, AX`	`97`	1	4

XLAT

Flags updated

None

Encoding

Table 114. XLAT encoding

Mnemonic	Opcode	Size	Latency
`xlat AL, BX`	`d7`	1	8

XOR

Flags updated

C	P	A	Z	S	T	I	D	O
M	M		M	M				M

Encoding

Table 115. XOR encoding

Mnemonic	Opcode	Size	Latency
`xor register8, register8`	`30 /r`	2-4	2
`xor memory8, register8`	`30 /r`	2-4	8
`xor register16, register16`	`31 /r`	2-4	2
`xor memory16, register16`	`31 /r`	2-4	8
`xor register8, register8`	`32 /r`	2-4	2
`xor register8, memory8`	`32 /r`	2-4	5
`xor register16, register16`	`33 /r`	2-4	2
`xor register16, memory16`	`33 /r`	2-4	5
`xor AL, immediate8`	`34 Ib`	2	2
`xor AX, immediate16`	`35 Iw`	3	2
`xor register8, immediate8`	`80 /6 Ib`	3-5	3
`xor memory8, immediate8`	`80 /6 Ib`	3-5	9
`xor register16, immediate16`	`81 /6 Iw`	4-6	3
`xor memory16, immediate16`	`81 /6 Iw`	4-6	9
`xor register8, immediate8`	`82 /6 Ib`	3-5	3
`xor memory8, immediate8`	`82 /6 Ib`	3-5	9
`xor register16, immediate8`	`83 /6 Ib`	3-5	4
`xor memory16, immediate8`	`83 /6 Ib`	3-5	11