# **Device name:**

Asus Vivobook X507ua



Specifications:

Brand: Asus

Series: Vivobook

Colour: Stary Gray

Form Factor: NoteBook

Item Height: 22 Millimeters

Item Width:36.5 Centimeters

Screen Size: 15.6 Inches

Maximum Display Resolution: 1920 x 1080 (Full HD)

Item Weight: 1.68 Kg

Product Dimensions : 26.6 x 36.5 x 2.2 cm

Batteries: 1 Lithium Polymer batteries required. (included)

Item model number: X507UF-EJ092T

Processor Brand: Intel

Processor Type: Core i5 8250U

Processor Speed: 3.40 GHz

Processor Count: 4

RAM Size: 8 GB

Memory Technology: DDR4

Computer Memory Type: GDDR4

Maximum Memory Supported: 16 GB

Hard Drive Size: 1024 GB

Hard Disk Technology: Mechanical Hard Drive

Hard Drive Interface: eSATA

Speaker Description: Built-in speaker Built-in microphone Sonic Master

Graphics Coprocessor: NVIDIA GeForce MX130

Graphics RAM Type: GDDR5

Graphics Card Ram Size: 2 GB

Graphics Card Interface: PCI-E

Connectivity Type: 802.11ac+Bluetooth 4.2 (Dual band) 2\*2

Number of USB 2.0 Ports: 2

Number of USB 3.0 Ports: 1

Number of HDMI Ports: 1

Number of Audio-out Ports: 1

Number of Microphone Ports: 1

Optical Drive Type: None

Hardware Platform: Windows

Operating System: Windows 10

Average Battery Life (in hours) :8 Hours

Lithium Battery Energy Content: 33 Watt Hours

Lithium battery Weight: 0.90 Grams

Number of Lithium Ion Cells: 3

# **Processor Architecture:**



Core i5-8250U is a 64-bit quad-core performance x86 mobile microprocessor introduced by Intel in mid-2017. This processor, which is based on an enhanced version of the Kaby Lake microarchitecture, is manufactured on Intel's 2nd generation enhanced 14nm+ process. The i5-8250U operates at 1.6 GHz with a TDP of 15 W and Turbo Boost frequency of up to 3.4 GHz. This MPU supports up to 32 GiB of dual-channel DDR4-2400 memory and incorporates Intel's UHD Graphics 620 IGP operating at 300 MHz with a burst frequency of 1.15 GHz.

**Memory organization:**

**System Memory Interface**

• LPDDR3 down or DDR3L/DDR3L-RS Non-ECC Unbuffered Small Outline Dual In-

Line Memory Modules with a maximum of one DIMM per channel or down

• LPDDR3 memory I/O Voltage of 1.2V. DDR3L/DDR3L-RS I/O Voltage of 1.35V

• Two memory channels. Single-channel and dual-channel memory organization

modes

• 64-bit wide channels

• Data burst length of eight for all memory organization modes

• Theoretical maximum memory bandwidth of:

— 21.3 GB/s in dual-channel mode assuming 1333 MT/s

— 25.6 GB/s in dual-channel mode assuming 1600 MT/s

— 29.8 GB/s in dual-channel mode assuming 1866 MT/s

**System Memory Technology Supported**

The Integrated Memory Controller (IMC) supports DDR3L/DDR3L-RS and LPDDR3

protocols with two independent, 64-bit wide channels. It supports unbuffered non-ECC

memory per channel, allowing up to two device ranks per channel.

Intel® Fast Memory Access (Intel® FMA)

Just-in-Time Command Scheduling

The system memory controller has an advanced command scheduler where all

pending requests are examined simultaneously to determine the most efficient request

to be issued next. The most efficient request is picked from all pending requests and

issued to system memory Just-in-Time to make optimal use of Command Overlapping.

Thus, instead of having all memory access requests go individually through an

arbitration mechanism forcing requests to be executed one at a time, the requests can

be started without interfering with the current request, allowing for concurrent issuing

of requests. This allows for optimized bandwidth and reduced latency while

maintaining appropriate command spacing to meet system memory protocol.

Command Overlap

Command Overlap allows the insertion of the DRAM commands between the Activate,

Pre-charge, and Read/Write commands normally used, as long as the inserted

commands do not affect the currently executing command. Multiple commands can be

issued in an overlapping manner, increasing the efficiency of system memory protocol.

Out-of-Order Scheduling

While leveraging the Just-in-Time Scheduling and Command Overlap enhancements,

the system memory controller continuously monitors pending requests to system

memory for the best use of bandwidth and reduction of latency. If there are multiple

requests to the same open page, these requests would be launched in a back-to-back

manner to make optimum use of the open memory page. This ability to reorder

requests on the fly allows the system memory controller to further reduce latency and

increase bandwidth efficiency.

**System Memory Frequency**

In all modes, the frequency of system memory is the lowest frequency of all memory

placed in the system, as determined through the SPD registers for the memory.

For systems using DDR3L/DDR3L-RS SO-DIMM modules with different latency

populated across the channels, the BIOS will use the slower of the two latencies for

both channels. For dual-channel mode, both channels must have the SO-DIMM

connector populated. For single-channel mode, only a single channel can have the SO-

DIMM connector be populated.

**System Memory Organization Modes**

The system memory controller supports two memory organization modes – single-

channel and dual-channel. Depending on how the DIMM Modules or DRAM Down

Devices are configured in each memory channel, a number of different configurations

can exist.

Single-Channel Mode

In this mode, all memory cycles are directed to a single-channel. Single-channel mode

is used when either Channel A or Channel B are populated in any order, but not both.

Dual-Channel Mode – Intel® Flex Memory Technology Mode

The system memory controller supportsIntelFlexMemory Technology Mode where

memory is divided into a symmetric and asymmetric zone. The symmetric zone starts

at the lowest address in each channel and is contiguous until the asymmetric zone

begins or until the top address of the channel with the smaller capacity is reached. In

this mode, the system runs with one zone of dual-channel mode and one zone of

single-channel mode, simultaneously, across the entire memory array. This mode is

used when both Channel A and Channel B are populated with memory but the total

amount of memory in each channel is not the same.

Note: Channels A and B can be mapped for Physical Channel 0 and 1 respectively or vice

versa; however, the Channel A size must be greater or equal to the Channel B size.

**INSTRUCTIONS:**

* ISA: Everything up to [AES](https://en.wikichip.org/w/index.php?title=x86/aes&action=edit&redlink=1) ([SMM](https://en.wikichip.org/w/index.php?title=x86/smm&action=edit&redlink=1), [FPU](https://en.wikichip.org/w/index.php?title=x86/fpu&action=edit&redlink=1), [MMX](https://en.wikichip.org/w/index.php?title=x86/mmx&action=edit&redlink=1), [SSE](https://en.wikichip.org/w/index.php?title=x86/sse&action=edit&redlink=1), [SSE2](https://en.wikichip.org/w/index.php?title=x86/sse2&action=edit&redlink=1), [SSE3](https://en.wikichip.org/w/index.php?title=x86/sse3&action=edit&redlink=1), [SSSE3](https://en.wikichip.org/w/index.php?title=x86/ssse3&action=edit&redlink=1), [SSE4.1](https://en.wikichip.org/w/index.php?title=x86/sse4.1&action=edit&redlink=1), [SSE4.2](https://en.wikichip.org/w/index.php?title=x86/sse4.2&action=edit&redlink=1), and [AES](https://en.wikichip.org/w/index.php?title=x86/aes&action=edit&redlink=1))

**1.AES**

**Instructions**

|  |  |
| --- | --- |
| **Instruction** | **Description** |
| **AESENC** | **Perform one round of an AES encryption flow** |
| **AESENCLAST** | **Perform the last round of an AES encryption flow** |
| **AESDEC** | **Perform one round of an AES decryption flow** |
| **AESDECLAST** | **Perform the last round of an AES decryption flow** |
| **AESKEYGENASSIST** | **Assist in AES round key generation** |
| **AESIMC** | **Assist in AES Inverse Mix Columns** |
| **PCLMULQDQ** | **Carryless multiply (**[**CLMUL**](https://en.wikipedia.org/wiki/CLMUL_instruction_set)**)** |

**2.SSM**

[ldc](https://www.staff.science.uu.nl/~dijks106/SSM/ssmtopics.html#ldc), load a constant.

[lds](https://www.staff.science.uu.nl/~dijks106/SSM/ssmtopics.html#lds), load a value relative to the SP.

[ldh](https://www.staff.science.uu.nl/~dijks106/SSM/ssmtopics.html#ldh), load a value relative to the HP.

[ldl](https://www.staff.science.uu.nl/~dijks106/SSM/ssmtopics.html#ldl), load a value relative to the MP.

[lda](https://www.staff.science.uu.nl/~dijks106/SSM/ssmtopics.html#lda), load a value pointed to by the value on top of the stack.

[ldr](https://www.staff.science.uu.nl/~dijks106/SSM/ssmtopics.html#ldr), load a register value.

[ldrr](https://www.staff.science.uu.nl/~dijks106/SSM/ssmtopics.html#ldrr), load a register with a value from another register.

[ldsa](https://www.staff.science.uu.nl/~dijks106/SSM/ssmtopics.html#ldsa), load address of value relative to the SP.

[ldla](https://www.staff.science.uu.nl/~dijks106/SSM/ssmtopics.html#ldla), load address of value relative to the MP.

[ldaa](https://www.staff.science.uu.nl/~dijks106/SSM/ssmtopics.html#ldaa), load address of value relative to the address on top of the stack.

[sts](https://www.staff.science.uu.nl/~dijks106/SSM/ssmtopics.html#sts), store a value relative to the SP.

[sth](https://www.staff.science.uu.nl/~dijks106/SSM/ssmtopics.html#sth), store a value relative to the HP.

[stl](https://www.staff.science.uu.nl/~dijks106/SSM/ssmtopics.html#stl), store a value relative to the MP

[sta](https://www.staff.science.uu.nl/~dijks106/SSM/ssmtopics.html#sta), store a value pointed to by a value on the stack

[str](https://www.staff.science.uu.nl/~dijks106/SSM/ssmtopics.html#str), store a value in a register.

[ajs](https://www.staff.science.uu.nl/~dijks106/SSM/ssmtopics.html#ajs), adjust the SP.

[link](https://www.staff.science.uu.nl/~dijks106/SSM/ssmtopics.html#link), save MP, adjust MP and SP suitable for programming language function entry.

[unlink](https://www.staff.science.uu.nl/~dijks106/SSM/ssmtopics.html#unlink), reverse of link

[beq](https://www.staff.science.uu.nl/~dijks106/SSM/ssmtopics.html#beq), [bne](https://www.staff.science.uu.nl/~dijks106/SSM/ssmtopics.html" \l "bne), [blt](https://www.staff.science.uu.nl/~dijks106/SSM/ssmtopics.html" \l "blt), [bgt](https://www.staff.science.uu.nl/~dijks106/SSM/ssmtopics.html" \l "bgt), [ble](https://www.staff.science.uu.nl/~dijks106/SSM/ssmtopics.html" \l "ble), [bge](https://www.staff.science.uu.nl/~dijks106/SSM/ssmtopics.html" \l "bge), respectively branching on equality, unequality, less than, greater than, less or equal, greater or equal. These instructions pop the stack, interpret it as a [condition code](https://www.staff.science.uu.nl/~dijks106/SSM/ssmtopics.html#cc) and jump accordingly.

[bra](https://www.staff.science.uu.nl/~dijks106/SSM/ssmtopics.html#bra), branch always, no popping of the stack.

[brf](https://www.staff.science.uu.nl/~dijks106/SSM/ssmtopics.html#brf) ([brt](https://www.staff.science.uu.nl/~dijks106/SSM/ssmtopics.html" \l "brt)), branch if top of stack is false (true).

[bsr](https://www.staff.science.uu.nl/~dijks106/SSM/ssmtopics.html#bsr), branch to subroutine. Like bra, but pushes the previous PC before jumping.

[jsr](https://www.staff.science.uu.nl/~dijks106/SSM/ssmtopics.html#jsr), jump to subroutine. Like bsr, but pops its destination from the stack

**3.FPU**

| **Instruction** | **Meaning** |
| --- | --- |
| **Integer Operands** | |
| [FILD](https://hjlebbink.github.io/x86doc/html/FILD.html) | Load integer |
| [FIST](https://hjlebbink.github.io/x86doc/html/FIST_FISTP.html) | Store integer |
| [FISTP](https://hjlebbink.github.io/x86doc/html/FIST_FISTP.html) | Store integer and pop |
| [FISTTP](https://hjlebbink.github.io/x86doc/html/FISTTP.html) | Behaves like the FISTP instruction but uses truncation, irrespective of the rounding mode specified in the floating-point control word (FCW) |
| **Binary-coded Decimal Operands** | |
| [FBLD](https://hjlebbink.github.io/x86doc/html/FBLD.html) | Load BCD |
| [FBSTP](https://hjlebbink.github.io/x86doc/html/FBSTP.html) | Store BCD and pop |
| **Floating-point Operands** | |
| [FLD](https://hjlebbink.github.io/x86doc/html/FLD.html) | Load floating-point value |
| [FST](https://hjlebbink.github.io/x86doc/html/FST_FSTP.html) | Store floating-point value |
| [FSTP](https://hjlebbink.github.io/x86doc/html/FST_FSTP.html) | Store floating-point value and pop |
| [FXCH](https://hjlebbink.github.io/x86doc/html/FXCH.html) | Exchange register |
| [FCMOVE](https://hjlebbink.github.io/x86doc/html/FCMOVcc.html) | Floating-point conditional move if equal |
| [FCMOVB](https://hjlebbink.github.io/x86doc/html/FCMOVcc.html) | Floating-point conditional move if below |
| [FCMOVBE](https://hjlebbink.github.io/x86doc/html/FCMOVcc.html) | Floating-point conditional move if below or equal |
| [FCMOVU](https://hjlebbink.github.io/x86doc/html/FCMOVcc.html) | Floating-point conditional move if unordered |
| [FCMOVNE](https://hjlebbink.github.io/x86doc/html/FCMOVcc.html) | Floating-point conditional move if not equal |
| [FCMOVNB](https://hjlebbink.github.io/x86doc/html/FCMOVcc.html) | Floating-point conditional move if not below |
| [FCMOVNBE](https://hjlebbink.github.io/x86doc/html/FCMOVcc.html) | Floating-point conditional move if not below or equal |
| [FCMOVNU](https://hjlebbink.github.io/x86doc/html/FCMOVcc.html) | Floating-point conditional move if not unordered |

**Basic Arithmetic Instructions**

| **Instruction** | **Meaning** |
| --- | --- |
| **Integer Operands** | |
| [FIADD](https://hjlebbink.github.io/x86doc/html/FADD_FADDP_FIADD.html) | Add integer |
| [FISUB](https://hjlebbink.github.io/x86doc/html/FSUB_FSUBP_FISUB.html) | Subtract integer |
| [FISUBR](https://hjlebbink.github.io/x86doc/html/FSUBR_FSUBRP_FISUBR.html) | Subtract integer reverse |
| [FIMUL](https://hjlebbink.github.io/x86doc/html/FMUL_FMULP_FIMUL.html) | Multiply integer |
| [FIDIV](https://hjlebbink.github.io/x86doc/html/FDIV_FDIVP_FIDIV.html) | Divide integer |
| [FIDIVR](https://hjlebbink.github.io/x86doc/html/FDIVR_FDIVRP_FIDIVR.html) | Divide integer reverse |
| **Floating-point Operands** | |
| [FADD](https://hjlebbink.github.io/x86doc/html/FADD_FADDP_FIADD.html) | Add floating-point |
| [FADDP](https://hjlebbink.github.io/x86doc/html/FADD_FADDP_FIADD.html) | Add floating-point and pop |
| [FSUB](https://hjlebbink.github.io/x86doc/html/FSUB_FSUBP_FISUB.html) | Subtract floating-point |
| [FSUBP](https://hjlebbink.github.io/x86doc/html/FSUB_FSUBP_FISUB.html) | Subtract floating-point and pop |
| [FSUBR](https://hjlebbink.github.io/x86doc/html/FSUBR_FSUBRP_FISUBR.html) | Subtract floating-point reverse |
| [FSUBRP](https://hjlebbink.github.io/x86doc/html/FSUBR_FSUBRP_FISUBR.html) | Subtract floating-point reverse and pop |
| [FMUL](https://hjlebbink.github.io/x86doc/html/FMUL_FMULP_FIMUL.html) | Multiply floating-point |
| [FMULP](https://hjlebbink.github.io/x86doc/html/FMUL_FMULP_FIMUL.html) | Multiply floating-point and pop |
| [FDIV](https://hjlebbink.github.io/x86doc/html/FDIV_FDIVP_FIDIV.html) | Divide floating-point |
| [FDIVP](https://hjlebbink.github.io/x86doc/html/FDIV_FDIVP_FIDIV.html) | Divide floating-point and pop |
| [FDIVR](https://hjlebbink.github.io/x86doc/html/FDIVR_FDIVRP_FIDIVR.html) | Divide floating-point reverse |
| [FDIVRP](https://hjlebbink.github.io/x86doc/html/FDIVR_FDIVRP_FIDIVR.html) | Divide floating-point reverse and pop |
| [FPREM](https://hjlebbink.github.io/x86doc/html/FPREM.html) | Partial remainder |
| [FPREM1](https://hjlebbink.github.io/x86doc/html/FPREM1.html) | IEEE Partial remainder |
| [FRNDINT](https://hjlebbink.github.io/x86doc/html/FRNDINT.html) | Round to integer |
| [FABS](https://hjlebbink.github.io/x86doc/html/FABS.html) | Absolute value |
| [FCHS](https://hjlebbink.github.io/x86doc/html/FCHS.html) | Change sign |
| [FSQRT](https://hjlebbink.github.io/x86doc/html/FSQRT.html) | Square root |
| [FSCALE](https://hjlebbink.github.io/x86doc/html/FSCALE.html) | Scale by power of two |
| [FXTRACT](https://hjlebbink.github.io/x86doc/html/FXTRACT.html) | Extract exponent and significant |

| **Instruction** | **Meaning** |
| --- | --- |
| [FSIN](https://hjlebbink.github.io/x86doc/html/FSIN.html) | Sine |
| [FCOS](https://hjlebbink.github.io/x86doc/html/FCOS.html) | Cosine |
| [FSINCOS](https://hjlebbink.github.io/x86doc/html/FSINCOS.html) | Sine and cosine |
| [FPTAN](https://hjlebbink.github.io/x86doc/html/FPTAN.html) | Partial tangent |
| [FPATAN](https://hjlebbink.github.io/x86doc/html/FPATAN.html) | Partial arctangent |
| [F2XM1](https://hjlebbink.github.io/x86doc/html/F2XM1.html) | 2x − 1 |
| [FYL2X](https://hjlebbink.github.io/x86doc/html/FYL2X.html) | y \* log2(x) |
| [FYL2XP1](https://hjlebbink.github.io/x86doc/html/FYL2XP1.html) | y \* log2(x+1) |

| **Instruction** | **Meaning** |
| --- | --- |
| [FLDZ](https://hjlebbink.github.io/x86doc/html/FLD1_FLDL2T_FLDL2E_FLDPI_FLDLG2_FLDLN2_FLDZ.html) | Load +0.0 |
| [FLD1](https://hjlebbink.github.io/x86doc/html/FLD1_FLDL2T_FLDL2E_FLDPI_FLDLG2_FLDLN2_FLDZ.html) | Load +1.0 |
| [FLDPI](https://hjlebbink.github.io/x86doc/html/FLD1_FLDL2T_FLDL2E_FLDPI_FLDLG2_FLDLN2_FLDZ.html) | Load Pi |
| [FLDL2E](https://hjlebbink.github.io/x86doc/html/FLD1_FLDL2T_FLDL2E_FLDPI_FLDLG2_FLDLN2_FLDZ.html) | Load log2(e) |
| [FLDLN2](https://hjlebbink.github.io/x86doc/html/FLD1_FLDL2T_FLDL2E_FLDPI_FLDLG2_FLDLN2_FLDZ.html) | Load loge(2) |
| [FLDL2T](https://hjlebbink.github.io/x86doc/html/FLD1_FLDL2T_FLDL2E_FLDPI_FLDLG2_FLDLN2_FLDZ.html) | Load log2(10) |
| [FLDLG2](https://hjlebbink.github.io/x86doc/html/FLD1_FLDL2T_FLDL2E_FLDPI_FLDLG2_FLDLN2_FLDZ.html) | Load log10(2) |

**Comparison Instructions**

| **Instruction** | **Meaning** |
| --- | --- |
| **Integer Operands** | |
| [FICOM](https://hjlebbink.github.io/x86doc/html/FICOM_FICOMP.html) | Compare integer |
| [FICOMP](https://hjlebbink.github.io/x86doc/html/FICOM_FICOMP.html) | Compare integer and pop |
| **Floating-point Operands** | |
| [FCOM](https://hjlebbink.github.io/x86doc/html/FCOM_FCOMP_FCOMPP.html) | Compare floating-point |
| [FCOMP](https://hjlebbink.github.io/x86doc/html/FCOM_FCOMP_FCOMPP.html) | Compare floating-point and pop |
| [FCOMPP](https://hjlebbink.github.io/x86doc/html/FCOM_FCOMP_FCOMPP.html) | Compare floating-point and pop twice |
| [FCOMI](https://hjlebbink.github.io/x86doc/html/FCOMI_FCOMIP_FUCOMI_FUCOMIP.html) | Compare floating-point and set EFLAGS |
| [FCOMIP](https://hjlebbink.github.io/x86doc/html/FCOMI_FCOMIP_FUCOMI_FUCOMIP.html) | Compare floating-point, set EFLAGS, and pop |
| [FUCOM](https://hjlebbink.github.io/x86doc/html/FUCOM_FUCOMP_FUCOMPP.html) | Unordered compare floating-point |
| [FUCOMP](https://hjlebbink.github.io/x86doc/html/FUCOM_FUCOMP_FUCOMPP.html) | Unordered compare floating-point and pop |
| [FUCOMPP](https://hjlebbink.github.io/x86doc/html/FUCOM_FUCOMP_FUCOMPP.html) | Unordered compare floating-point and pop twice |
| [FUCOMI](https://hjlebbink.github.io/x86doc/html/FCOMI_FCOMIP_FUCOMI_FUCOMIP.html) | Unordered compare floating-point and set EFLAGS |
| [FUCOMIP](https://hjlebbink.github.io/x86doc/html/FCOMI_FCOMIP_FUCOMI_FUCOMIP.html) | Unordered compare floating-point, set EFLAGS, and pop |
| [FTST](https://hjlebbink.github.io/x86doc/html/FTST.html) | Test floating-point (compare with 0.0) |
| [FXAM](https://hjlebbink.github.io/x86doc/html/FXAM.html) | Examine floating-point |

**Control Instructions**

| **Instruction** | **Meaning** |
| --- | --- |
| [FFREE](https://hjlebbink.github.io/x86doc/html/FFREE.html) | Free floating-point register |
| [FINCSTP](https://hjlebbink.github.io/x86doc/html/FINCSTP.html) | Increment FPU register stack pointer |
| [FDECSTP](https://hjlebbink.github.io/x86doc/html/FDECSTP.html) | Decrement FPU register stack pointer |
| [FINIT](https://hjlebbink.github.io/x86doc/html/FINIT_FNINIT.html) | Initialize FPU after checking error conditions |
| [FNINIT](https://hjlebbink.github.io/x86doc/html/FINIT_FNINIT.html) | Initialize FPU without checking error conditions |
| [FSTSW](https://hjlebbink.github.io/x86doc/html/FSTSW_FNSTSW.html) | Store FPU status word after checking error conditions |
| [FNSTSW](https://hjlebbink.github.io/x86doc/html/FSTSW_FNSTSW.html) | Store FPU status word without checking error conditions |
| [FCLEX](https://hjlebbink.github.io/x86doc/html/FCLEX_FNCLEX.html) | Clear floating-point exception flags after checking for error conditions |
| [FNCLEX](https://hjlebbink.github.io/x86doc/html/FCLEX_FNCLEX.html) | Clear floating-point exception flags without checking for error conditions |
| [FSTCW](https://hjlebbink.github.io/x86doc/html/FSTCW_FNSTCW.html) | Store FPU control word after checking error conditions |
| [FNSTCW](https://hjlebbink.github.io/x86doc/html/FSTCW_FNSTCW.html) | Store FPU control word without checking error conditions |
| [FLDCW](https://hjlebbink.github.io/x86doc/html/FLDCW.html) | Load FPU control word |
| [FSAVE](https://hjlebbink.github.io/x86doc/html/FSAVE_FNSAVE.html) | Save FPU state after checking error conditions |
| [FNSAVE](https://hjlebbink.github.io/x86doc/html/FSAVE_FNSAVE.html) | Save FPU state without checking error conditions |
| [FRSTOR](https://hjlebbink.github.io/x86doc/html/FRSTOR.html) | Restore FPU state |
| [FSTENV](https://hjlebbink.github.io/x86doc/html/FSTENV_FNSTENV.html) | Store FPU environment after checking error conditions |
| [FNSTENV](https://hjlebbink.github.io/x86doc/html/FSTENV_FNSTENV.html) | Store FPU environment without checking error conditions |
| [FLDENV](https://hjlebbink.github.io/x86doc/html/FLDENV.html) | Load FPU environment |
| [FNOP](https://hjlebbink.github.io/x86doc/html/FNOP.html) | FPU no operation |
| [FWAIT](https://hjlebbink.github.io/x86doc/html/WAIT_FWAIT.html) | Wait for FPU |

**FPU and SIMD State Management Instructions**

| **Instruction** | **Meaning** |
| --- | --- |
| [FXSAVE](https://hjlebbink.github.io/x86doc/html/FXSAVE.html) | Save x87 FPU and SIMD state |
| [FXRSTOR](https://hjlebbink.github.io/x86doc/html/FXRSTOR.html) | Restore x87 FPU and SIMD state |

**4.MMX**

movd (MOVe Doubleword) can move either a 32-bit register or memory location into or out of the bottom 32 bits of an MMX register. When data moves in, the top 32 bits of the MMX register are set to zero.  
  
movq (MOVe Quadword) moves 64-bit quantites between memory and an MMX register or between two MMX register.

**MMX — Boolean Logic**

MMX is integer-only, so it makes sense for it to offer normal boolean logic operations. These are fairly easy to grasp.  
  
pxor can exclusive-or (XOR) any two MMX registers, an MMX register and memory, or an MMX register and a constant.  
  
por can bitwise-or (OR) any two MMX registers, an MMX register and memory, or an MMX register and a constant.  
  
pand can bitwise-and (AND) any two MMX registers, an MMX register and memory, or an MMX register and a constant.  
  
pandn can bitwise-not-and (NAND) any two MMX registers, an MMX register and memory, or an MMX register and a constant.  
  
These instructions operate the same regardless of how the data is arranged in the register (whether it's a 64-bit value or 8 8-bit values). That's the nature of boolean logic, after all.  
  
There are also a number of shift operations available in MMX.  
  
psllw shifts a specified register left a certain number of bits, operating on words (16 bits).  
  
pslld shifts a specified register left a certain number of bits, operating on doublewords (32 bits).  
  
psllq shifts a specified register left a certain number of bits, operating on a quadwords (64 bits).  
  
psrlw shifts a specified register right a certain number of bits, operating on words (16 bits). This is a logical shift, not arithmetic.  
  
psrld shifts a specified register right a certain number of bits, operating on doublewords (32 bits). This is a logical shift, not arithmetic.  
  
psrlq shifts a specified register right a certain number of bits, operating on a quadwords (64 bits). This is a logical shift, not arithmetic.  
  
psraw shifts a specified register left a certain number of bits, operating on words (16 bits). This one is arithmetic, which means the new top bits are a copy of the original top bit (the sign bit).  
  
psrad shifts a specified register left a certain number of bits, operating on doublewords (32 bits). This one is also arithmetic.  
  
The shift operations *do* distinguish between the various sizes of the register. This is necessary to keep bits in one value from affecting adjacent values.

**MMX — Math**

MMX has a number of basic math operations included.  
  
paddb adds an MMX register and another MMX register or memory as unsigned 8-bit bytes.  
  
paddsb is just like paddb, except the bytes are signed and the values saturate instead of wrapping around. This instruction saturates at 127 (0x7f) or -128 (0x80).  
  
paddusb is like paddsb but with unsigned bytes. This instruction saturates at 255 (0xff).  
  
paddw add an MMX register and another register or memory as unsigned 16-bit words.  
  
paddsw is just like paddsb except it uses 16-bit words instead of 8-bit bytes. This instruction saturates at 32767 (0x7fff) or -32768 (0x8000).  
  
paddusw adds unsigned words, and saturates at 65535 (0xffff).  
  
paddd adds a register and another register or memory location as unsigned 32-bit doublewords.  
  
psubb subtracts a memory location or MMX register from another register, operating on unsigned 8-bit bytes.  
  
psubsb subtracts a register or memory location from another register, using signed bytes, and saturates at -128 (0x80) or 127 (0x7f).  
  
psubusb subtracts unsigned bytes with saturation, similar to psubsb. This instruction saturates at 0 (0x00).  
  
psubw subtracts unsigned 16-bit words.  
  
psubsw subtracts signed 16-bit words, and saturates at 32767 0x7fff) or -32768 (0x8000).  
  
psubusb subtracts unsigned 16-bit words, saturating at 0 (0x0000).  
  
psubd subtracts a register or memory location from another register using unsigned 32-bit doublewords.  
  
MMX also supplies a few multiply instructions.  
  
pmulhw multiplies a register with another register or memory locatioin using signed 16-bit words. It then stores the upper 16 bits of each 32-bit result.  
  
pmullw multiplies just like pmulhw except it stores the lower 16 bits of each 16-bit result.  
  
pmaddwd multiplies signed 16-bit words and adds the 32-bit results. It multiplies an MMX register and another register or memory location.

**MMX — Comparasons**

MMX provides a bunch of instructions for performing various-sized compares.  
  
pcmpeqb compares for 8-bit equality, between an MMX register and another register or memory location. For pairs that are equal the result is all ones (0xff), otherwise it is zero (0x00).  
  
pcmpgtb performs a 'Greater Than' 8-bit value compare in the same manner as pcmpeqb. For larger values, the result is all ones (0xff), otherwise zero (0x00).  
  
pcmpeqw compares 16-bit words for equality, just like pcmpeqb.  
  
pcmpgtw is the 16-bit equivalent of pcmpgtb.  
  
pcmpeqd compares 32-bit doublewords for equality.  
  
pcmpgtd compares magnitudes of 32-bit doublewords, just like pcmpgtw does for words.

**MMX — Data Packing**

There are several instructions used for data packing provded by MMX. These instructions generally sign- or zero-extend values, interleave values, and truncate values.  
  
packssdw takes a register and another register or memory location, and saturates the 32-bit doublewords into 16-bit words. The doublewords and word results are signed.  
  
packsswb takes a register and another register or memory location, and saturates the 16-bit signed words from both into signed 8-bit bytes in the first register.  
  
packuswb is similar to packsswb, with the source words and resultant bytes being unsigned instead of signed.  
  
punpckhbw unpacks the top 32 bits of 2 MMX registers or a register and a memory location into a destination MMX register. The data is interleaved in 8-bit pieces, with the 2nd operand going to the top halves and the 1st operand going to the bottom halves.  
  
punpckhdq is similar to punpckhbw, except it uses 32-bit pieces instead of 8-bit ones. The 1st operand's top half goes to the bottom half of the destination, and the 2nd operand's top goes to the top half.  
  
punpckhwd is also similar to punpckhbw, but with 16-bit pieces instead of 8-bit ones. The 1st operand goes to the top halves, and the 2nd operand goes to the bottom halves.  
  
punpcklbw is like punpckhbw, but instead of taking data from the top half of the sources, data is taken from the bottom.  
  
punpckldq is like punpckhdq, but it uses the bottom 32-bit pieces instead of the top 32-bit pieces.  
  
punpcklwd is like punpckhwd, but it uses the bottoms of the sources instead of the tops.

**5.SSE**

addss - Adds the lowest single-precision values, top 3 remain unchanged.  
subps - Subtracts 4 single-precision floating-point values from 4 other single-precision floating-point values.  
subss - Subtracts the lowest single-precision values, top 3 remain unchanged.  
mulps - Multiplies 4 single-precision floating-point values with 4 other single-precision values.  
mulss - Multiplies the lowest single-precision values, top 3 remain unchanged.  
divps - Divides 4 single-precision floating-point values by 4 other single-precision floating-point values.  
divss - Divides the lowest single-precision values, top 3 remain unchanged.  
rcpps - Reciprocates (1/x) 4 single-precision floating-point values.  
rcpss - Reciprocates the lowest single-precision values, top 3 remain unchanged.  
sqrtps - Square root of 4 single-precision values.  
sqrtss - Square root of lowest value, top 3 remain unchanged.  
rsqrtps - Reciprocal square root of 4 single-precision floating-point values.  
rsqrtss - Reciprocal square root of lowest single-precision value, top 3 remain unchanged.  
maxps - Returns maximum of 2 values in each of 4 single-precision values.  
maxss - Returns maximum of 2 values in the lowest single-precision value. Top 3 remain unchanged.  
minps - Returns minimum of 2 values in each of 4 single-precision values.  
minss - Returns minimum of 2 values in the lowest single-precision value, top 3 remain unchanged.  
pavgb - Returns average of 2 values in each of 8 bytes.  
pavgw - Returns average of 2 values in each of 4 words.  
psadbw - Returns sum of absolute differences of 8 8bit values. Result in bottom 16 bits.  
pextrw - Extracts 1 of 4 words.  
pinsrw - Inserts 1 of 4 words.  
pmaxsw - Returns maximum of 2 values in each of 4 signed word values.  
pmaxub - Returns maximum of 2 values in each of 8 unsigned byte values.  
pminsw - Returns minimum of 2 values in each of 4 signed word values.  
pminub - Returns minimum of 2 values in each of 8 unsigned byte values.  
pmovmskb - builds mask byte from top bit of 8 byte values.  
pmulhuw - Multiplies 4 unsigned word values and stores the high 16bit result.  
pshufw - Shuffles 4 word values. Takes 2 128bit values (source and dest) and an 8-bit immediate value, and then fills in each Dest 32-bit value from a Source 32-bit value specified by the immediate. The immediate byte is broken into 4 2-bit values.  
  
Logic:  
andnps - Logically ANDs 4 single-precision values with the logical inverse (NOT) of 4 other single-precision values.  
andps - Logically ANDs 4 single-precision values with 4 other single-precision values.  
orps - Logically ORs 4 single-precision values with 4 other single-precision values.  
xorps - Logically XORs 4 single-precision values with 4 other single-precision values.  
  
Compare:  
cmpxxps - Compares 4 single-precision values.  
cmpxxss - Compares lowest 2 single-precision values.  
comiss - Compares lowest 2 single-recision values and stores result in EFLAGS.  
ucomiss - Compares lowest 2 single-precision values and stores result in EFLAGS. (QNaNs don't throw exceptions with ucomiss, unlike comiss.)  
Compare Codes (the xx parts above):  
eq - Equal to.  
lt - Less than.  
le - Less than or equal to.  
ne - Not equal.  
nlt - Not less than.  
nle - Not less than or equal to.  
ord - Ordered.  
unord - Unordered.  
  
Conversion:  
cvtpi2ps - Converts 2 32bit integers to 32bit floating-point values. Top 2 values remain unchanged.  
cvtps2pi - Converts 2 32bit floating-point values to 32bit integers.  
cvtsi2ss - Converts 1 32bit integer to 32bit floating-point value. Top 3 values remain unchanged.  
cvtss2si - Converts 1 32bit floating-point value to 32bit integer.  
cvttps2pi - Converts 2 32bit floating-point values to 32bit integers using truncation.  
cvttss2si - Converts 1 32bit floating-point value to 32bit integer using truncation.  
  
State:  
fxrstor - Restores FP and SSE State.  
fxsave - Stores FP and SSE State.  
ldmxcsr - Loads the mxcsr register.  
stmxcsr - Stores the mxcsr register.  
  
Load/Store:  
movaps - Moves a 128bit value.  
movhlps - Moves high half to a low half.  
movlhps - Moves low half to upper halves.?  
movhps - Moves 64bit value into top half of an xmm register.  
movlps - Moves 64bit value into bottom half of an xmm register.  
movmskps - Moves top bits of single-precision values into bottom 4 bits of a 32bit register.  
movss - Moves the bottom single-precision value, top 3 remain unchanged if the destination is another xmm register, otherwise they're set to zero.  
movups - Moves a 128bit value. Address can be unaligned.  
maskmovq - Moves a 64bit value according to a mask.  
movntps - Moves a 128bit value directly to memory, skipping the cache. (NT stands for "Non Temporal".)  
movntq - Moves a 64bit value directly to memory, skipping the cache.  
  
Shuffling:  
shufps - Shuffles 4 single-precision values. Complex.  
unpckhps - Unpacks single-precision values from high halves.  
unpcklps - Unpacks single-precision values from low halves.  
  
Cache Control:  
prefetchT0 - Fetches a cache-line of data into all levels of cache.  
prefetchT1 - Fetches a cache-line of data into all but the highest levels of cache.  
prefetchT2 - Fetches a cache-line of data into all but the two highest levels of cache.  
prefetchNTA - Fetches data into only the highest level of cache, not the lower levels.  
sfence - Guarantees that all memory writes issued before the sfence instruction are completed before any writes after the sfence instruction.

6.SSE2

Arithmetic:  
addpd - Adds 2 64bit doubles.  
addsd - Adds bottom 64bit doubles.  
subpd - Subtracts 2 64bit doubles.  
subsd - Subtracts bottom 64bit doubles.  
mulpd - Multiplies 2 64bit doubles.  
mulsd - Multiplies bottom 64bit doubles.  
divpd - Divides 2 64bit doubles.  
divsd - Divides bottom 64bit doubles.  
maxpd - Gets largest of 2 64bit doubles for 2 sets.  
maxsd - Gets largets of 2 64bit doubles to bottom set.  
minpd - Gets smallest of 2 64bit doubles for 2 sets.  
minsd - Gets smallest of 2 64bit values for bottom set.  
paddb - Adds 16 8bit integers.  
paddw - Adds 8 16bit integers.  
paddd - Adds 4 32bit integers.  
paddq - Adds 2 64bit integers.  
paddsb - Adds 16 8bit integers with saturation.  
paddsw - Adds 8 16bit integers using saturation.  
paddusb - Adds 16 8bit unsigned integers using saturation.  
paddusw - Adds 8 16bit unsigned integers using saturation.  
psubb - Subtracts 16 8bit integers.  
psubw - Subtracts 8 16bit integers.  
psubd - Subtracts 4 32bit integers.  
psubq - Subtracts 2 64bit integers.  
psubsb - Subtracts 16 8bit integers using saturation.  
psubsw - Subtracts 8 16bit integers using saturation.  
psubusb - Subtracts 16 8bit unsigned integers using saturation.  
psubusw - Subtracts 8 16bit unsigned integers using saturation.  
pmaddwd - Multiplies 16bit integers into 32bit results and adds results.  
pmulhw - Multiplies 16bit integers and returns the high 16bits of the result.  
pmullw - Multiplies 16bit integers and returns the low 16bits of the result.  
pmuludq - Multiplies 2 32bit pairs and stores 2 64bit results.  
rcpps - Approximates the reciprocal of 4 32bit singles.  
rcpss - Approximates the reciprocal of bottom 32bit single.  
sqrtpd - Returns square root of 2 64bit doubles.  
sqrtsd - Returns square root of bottom 64bit double.  
  
Logic:  
andnpd - Logically NOT ANDs 2 64bit doubles.  
andnps - Logically NOT ANDs 4 32bit singles.  
andpd - Logically ANDs 2 64bit doubles.  
pand - Logically ANDs 2 128bit registers.  
pandn - Logically Inverts the first 128bit operand and ANDs with the second.  
por - Logically ORs 2 128bit registers.  
pslldq - Logically left shifts 1 128bit value.  
psllq - Logically left shifts 2 64bit values.  
pslld - Logically left shifts 4 32bit values.  
psllw - Logically left shifts 8 16bit values.  
psrad - Arithmetically right shifts 4 32bit values.  
psraw - Arithmetically right shifts 8 16bit values.  
psrldq - Logically right shifts 1 128bit values.  
psrlq - Logically right shifts 2 64bit values.  
psrld - Logically right shifts 4 32bit values.  
psrlw - Logically right shifts 8 16bit values.  
pxor - Logically XORs 2 128bit registers.  
orpd - Logically ORs 2 64bit doubles.  
xorpd - Logically XORs 2 64bit doubles.  
  
Compare:  
cmppd - Compares 2 pairs of 64bit doubles.   
cmpsd - Compares bottom 64bit doubles.  
comisd - Compares bottom 64bit doubles and stores result in EFLAGS.  
ucomisd - Compares bottom 64bit doubles and stores result in EFLAGS. (QNaNs don't throw exceptions with ucomisd, unlike comisd.  
pcmpxxb - Compares 16 8bit integers.  
pcmpxxw - Compares 8 16bit integers.  
pcmpxxd - Compares 4 32bit integers.  
Compare Codes (the xx parts above):  
eq - Equal to.  
lt - Less than.  
le - Less than or equal to.  
ne - Not equal.  
nlt - Not less than.  
nle - Not less than or equal to.  
ord - Ordered.  
unord - Unordered.  
  
Conversion:  
cvtdq2pd - Converts 2 32bit integers into 2 64bit doubles.  
cvtdq2ps - Converts 4 32bit integers into 4 32bit singles.  
cvtpd2pi - Converts 2 64bit doubles into 2 32bit integers in an MMX register.  
cvtpd2dq - Converts 2 64bit doubles into 2 32bit integers in the bottom of an XMM register.  
cvtpd2ps - Converts 2 64bit doubles into 2 32bit singles in the bottom of an XMM register.  
cvtpi2pd - Converts 2 32bit integers into 2 32bit singles in the bottom of an XMM register.  
cvtps2dq - Converts 4 32bit singles into 4 32bit integers.  
cvtps2pd - Converts 2 32bit singles into 2 64bit doubles.  
cvtsd2si - Converts 1 64bit double to a 32bit integer in a GPR.  
cvtsd2ss - Converts bottom 64bit double to a bottom 32bit single. Tops are unchanged.  
cvtsi2sd - Converts a 32bit integer to the bottom 64bit double.  
cvtsi2ss - Converts a 32bit integer to the bottom 32bit single.  
cvtss2sd - Converts bottom 32bit single to bottom 64bit double.  
cvtss2si - Converts bottom 32bit single to a 32bit integer in a GPR.  
cvttpd2pi - Converts 2 64bit doubles to 2 32bit integers using truncation into an MMX register.  
cvttpd2dq - Converts 2 64bit doubles to 2 32bit integers using truncation.  
cvttps2dq - Converts 4 32bit singles to 4 32bit integers using truncation.  
cvttps2pi - Converts 2 32bit singles to 2 32bit integers using truncation into an MMX register.  
cvttsd2si - Converts a 64bit double to a 32bit integer using truncation into a GPR.  
cvttss2si - Converts a 32bit single to a 32bit integer using truncation into a GPR.  
  
Load/Store:  
(is "minimize cache pollution" the same as "without using cache"??)  
movq - Moves a 64bit value, clearing the top 64bits of an XMM register.  
movsd - Moves a 64bit double, leaving tops unchanged if move is between two XMMregisters.  
movapd - Moves 2 aligned 64bit doubles.  
movupd - Moves 2 unaligned 64bit doubles.  
movhpd - Moves top 64bit value to or from an XMM register.  
movlpd - Moves bottom 64bit value to or from an XMM register.  
movdq2q - Moves bottom 64bit value into an MMX register.  
movq2dq - Moves an MMX register value to the bottom of an XMM register. Top is cleared to zero.  
movntpd - Moves a 128bit value to memory without using the cache. NT is "Non Temporal."  
movntdq - Moves a 128bit value to memory without using the cache.  
movnti - Moves a 32bit value without using the cache.  
maskmovdqu - Moves 16 bytes based on sign bits of another XMM register.  
pmovmskb - Generates a 16bit Mask from the sign bits of each byte in an XMM register.  
  
Shuffling:  
pshufd - Shuffles 32bit values in a complex way.  
pshufhw - Shuffles high 16bit values in a complex way.  
pshuflw - Shuffles low 16bit values in a complex way.  
unpckhpd - Unpacks and interleaves top 64bit doubles from 2 128bit sources into 1.  
unpcklpd - Unpacks and interleaves bottom 64bit doubles from 2 128 bit sources into 1.  
punpckhbw - Unpacks and interleaves top 8 8bit integers from 2 128bit sources into 1.  
punpckhwd - Unpacks and interleaves top 4 16bit integers from 2 128bit sources into 1.  
punpckhdq - Unpacks and interleaves top 2 32bit integers from 2 128bit sources into 1.  
punpckhqdq - Unpacks and interleaces top 64bit integers from 2 128bit sources into 1.  
punpcklbw - Unpacks and interleaves bottom 8 8bit integers from 2 128bit sources into 1.  
punpcklwd - Unpacks and interleaves bottom 4 16bit integers from 2 128bit sources into 1.  
punpckldq - Unpacks and interleaves bottom 2 32bit integers from 2 128bit sources into 1.  
punpcklqdq - Unpacks and interleaces bottom 64bit integers from 2 128bit sources into 1.  
packssdw - Packs 32bit integers to 16bit integers using saturation.  
packsswb - Packs 16bit integers to 8bit integers using saturation.  
packuswb - Packs 16bit integers to 8bit unsigned integers unsing saturation.  
  
Cache Control:  
clflush - Flushes a Cache Line from all levels of cache.  
lfence - Guarantees that all memory loads issued before the lfence instruction are completed before anyloads after the lfence instruction.  
mfence - Guarantees that all memory reads and writes issued before the mfence instruction are completed before any reads or writes after the mfence instruction.  
pause - Pauses execution for a set amount of time.

7.SSE3

Arithmetic:  
addsubpd - Adds the top two doubles and subtracts the bottom two.  
addsubps - Adds top singles and subtracts bottom singles.  
haddpd - Top double is sum of top and bottom, bottom double is sum of second operand's top and bottom.  
haddps - Horizontal addition of single-precision values.  
hsubpd - Horizontal subtraction of double-precision values.  
hsubps - Horizontal subtraction of single-precision values.  
  
Load/Store:  
lddqu - Loads an unaligned 128bit value.

movddup - Loads 64bits and duplicates it in the top and bottom halves of a 128bit register.  
movshdup - Duplicates the high singles into high and low singles.  
movsldup - Duplicates the low singles into high and low singles.  
fisttp - Converts a floating-point value to an integer using truncation.  
  
Process Control:  
monitor - Sets up a region to monitor for activity.  
mwait - Waits until activity happens in a region specified by monitor.

8.SSSE3

psignd - Gives 32bit integer magnitudes the sign of the 2nd operand.  
psignw - Gives 16bit integer magnitudes the sign of the 2nd operand.  
psignb - Gives 8bit integer magnitudes the sign of the 2nd operand.  
phaddd - Horizontal addition of unsigned 32bit integers.  
phaddw - Horizontal addition of unsigned 16bit integers.  
phaddsw - Horizontal saturated addition of 16bit integers.  
phsubd - Horizontal subtraction of unsigned 32bit integers.  
phsubw - Horizontal subtraction of unsigned 16bit integers.  
phsubsw - Horizontal saturated subtraction of 16bit words.  
pmaddubsw - Multiply-accumulate instruction (finally).  
pabsd - abs() for 32bit integers.  
pabsw - abs() for 16bit integers.  
pabsb - abs() for 8bit integers.  
pmulhrsw - 16bit integer multiplication, stores top 16bits of result.  
pshufb - Another complex shuffle instruction.  
palignr - Combines two register values, and extracts a register-width value from it, based on an offset.

**9.SSSE4.1&4.2**

SSE4.1  
mpsadbw - Sum of absolute differences.  
phminposuw - minimum+index extraction (16bit word).  
pmuldq - packed multiply.  
pmulld - packed multiply.  
dpps - dot product, single precision.  
dppd - dot product, double precision.  
blendps - conditional copy.  
blendpd - conditional copy.  
blendvps - conditional copy.  
blendvpd - conditional copy.  
pblendvb - conditional copy.  
pblendw - conditional copy.  
pminsb - packed minimum signed byte.  
pmaxsb - packed maximum signed byte.  
pminuw - packed minimum unsigned word.  
pmaxuw - packed maximum unsigned word.  
pminud - packed minimum unsigned dword.  
pmaxud - packed maximum unsigned dword.  
pminsd - packed minimum signed dword.  
pmaxsd - packed maximum signed dword.  
roundps - packed round single precision float to integer.  
roundss - scalar round single precision float to integer.  
roundpd - packed round double precision float to integer.  
roundsd - scalar round double precision float to integer.  
insertps - complex data shuffling.  
pinsrb - complex data shuffling.  
pinsrd - complex data shuffling.  
pinsrq - complex data shuffling.  
extractps - complex data shuffling.  
pextrb - complex data shuffling.  
pextrw - complex data shuffling.  
pextrd - complex data shuffling.  
pextrq - complex data shuffling.  
pmovsxbw - packed sign extension.  
pmovzxbw - packed zero extension.  
pmovsxbd - packed sign extension.  
pmovzxbd - packed zero extension.  
pmovsxbq - packed sign extension.  
pmovzxbq - packed zero extension.  
pmovxswd - packed sign extension.  
pmovzxwd - packed zero extension.  
pmovsxwq - packed sign extension.  
pmovzxwq - packed zero extension.  
pmovsxdq - packed sign extension.  
pmovzxdq - packed zero extension.  
ptest - same as test, but for sse registers.  
pcmpeqq - quadword compare for equality.  
packusdw - saturating signed dwords to unsigned words.  
movntdqa - Non-temporal aligned move (this uses write-combining for efficiency).  
  
SSE4.2  
crc32 - CRC32C function (using 0x11edc6f41 as the polynomial).  
pcmpestri - Packed compare explicit length string, Index.  
pcmpestrm - Packed compare explicit length string, Mask.  
pcmpistri - Packed compare implicit length string, Index.  
pcmpistrm - Packed compare implicit length string, Mask.  
pcmpgtq - Packed compare, greater than.  
popcnt - Population count.

I/O componenets:

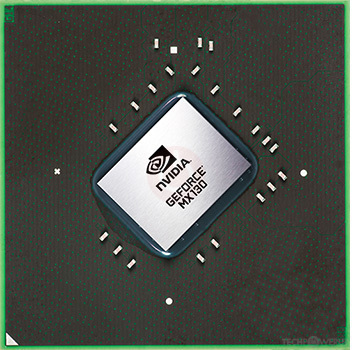
there two USB 2.0 ports and one USB 3.0 port

one hdmi one earphone jack

one chargeing port.

USB 2.0 USB 3.0 hdmi

graphics card:

****

**NVIDIA GeForce MX130 brings amazing performance to everyday laptops. Tap into the powerful NVIDIA Maxwel architecture for fast, smooth HD photo and video editing, plus better gaming. It also works seamlessly with the power-saving NVIDIA Optimus technology to let you do a whole lot more between charges.**