**ORGANISASI DAN ARSITEKTUR KOMPUTER**

**LATIFAH HANIF**

**L200150038**

1. Instruction set pada prosesor AMD.

| **Instruction set** | **Bits** | **Version** | **Introduced** | **Max #** [**operands**](https://en.wikipedia.org/wiki/Operand) | **Type** | **Design** | [**Registers**](https://en.wikipedia.org/wiki/Processor_register) **(excluding FP/vector)** | **Instruction encoding** | [**Branch**](https://en.wikipedia.org/wiki/Branch_%28computer_science%29) **evaluation** | [**Endianness**](https://en.wikipedia.org/wiki/Endianness) | **Extensions** | **Open** | **Royalty free** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| [6502](https://en.wikipedia.org/wiki/MOS_Technology_6502) | 8 |  | 1975 | 1 | Register Memory | CISC | 3 | Variable (8- to 32-bit) | Condition register | Little |  |  |  |
| [65k](https://en.wikipedia.org/w/index.php?title=65k&action=edit&redlink=1) | 64 (8→64)[[1]](https://en.wikipedia.org/wiki/Comparison_of_instruction_set_architectures#cite_note-65k_arch-1) |  | 2006? | 1 | Memory Memory[[*citation needed*](https://en.wikipedia.org/wiki/Wikipedia:Citation_needed)] | CISC | 1 | Variable (8-bit to 256 bytes) | Compare and branch[[*citation needed*](https://en.wikipedia.org/wiki/Wikipedia:Citation_needed)] | Little |  |  |  |
| 68000 / [680x0](https://en.wikipedia.org/wiki/Motorola_68000_series) | 32 |  | 1979 | 2 | Register Memory | [CISC](https://en.wikipedia.org/wiki/Complex_instruction_set_computer) | 8 data and 8 address | Variable | Condition register | Big |  | Unknown | Unknown |
| [8080](https://en.wikipedia.org/wiki/8080) | 8 |  | 1974 | 2 | Register Memory | CISC | 8 | Variable (8 to 24 bits) | Condition register | Little |  |  |  |
| [8051](https://en.wikipedia.org/wiki/8051) | 32 (8→32) |  | 1977? | 1 | Register Register | CISC | * 32 in 4-bit * 16 in 8-bit * 8 in 16-bit * 4 in 32-bit | Variable (8-bit to 128 bytes) | Compare and branch | Little |  |  |  |
| 8086 / [x86](https://en.wikipedia.org/wiki/X86) | 16, 32, 64 (16→32→64) |  | 1978 | 2 (integer) 3 ([AVX](https://en.wikipedia.org/wiki/Advanced_Vector_Extensions))[[2]](https://en.wikipedia.org/wiki/Comparison_of_instruction_set_architectures#cite_note-2) | Register Memory | CISC | * 8 (+ 4 or 6 segment registers) in 16/32-bit * 16 (+ 2 segment register gs/cs in 64-bit | Variable (8086: 8- to 48-bit) | Condition code | Little | [x87](https://en.wikipedia.org/wiki/X87), [IA-32](https://en.wikipedia.org/wiki/IA-32), [MMX](https://en.wikipedia.org/wiki/MMX_%28instruction_set%29), [3DNow!](https://en.wikipedia.org/wiki/3DNow%21), [SSE](https://en.wikipedia.org/wiki/Streaming_SIMD_Extensions), [SSE2](https://en.wikipedia.org/wiki/SSE2), [PAE](https://en.wikipedia.org/wiki/Physical_Address_Extension), [x86-64](https://en.wikipedia.org/wiki/X86-64), [SSE3](https://en.wikipedia.org/wiki/SSE3), [SSE4](https://en.wikipedia.org/wiki/SSE4), [SSE5](https://en.wikipedia.org/wiki/SSE5), [AVX](https://en.wikipedia.org/wiki/Advanced_Vector_Extensions), [AES](https://en.wikipedia.org/wiki/AES_instruction_set), [FMA](https://en.wikipedia.org/wiki/FMA_instruction_set) | No | No |
| [Alpha](https://en.wikipedia.org/wiki/DEC_Alpha) | 64 |  | 1992 | 3 | Register Register | [RISC](https://en.wikipedia.org/wiki/Reduced_instruction_set_computer) | 32 (including "zero") | Fixed (32-bit) | Condition register | Bi | MVI, BWX, FIX, CIX | No | Unknown |
| [ARM](https://en.wikipedia.org/wiki/ARM_architecture) | 32/16 | ARMv7 and earlier | 1983 | 3 | Register Register | RISC | * 7 in 16-bit thumb mode * 14 in 32-bit * + direct link register (not general purpose) | Fixed (32-bit), Thumb: Fixed (16-bit), Thumb-2: Variable (16- and 32-bit) | Condition code | Bi | NEON, [Jazelle](https://en.wikipedia.org/wiki/Jazelle), VFP, [TrustZone](https://en.wikipedia.org/wiki/ARM_architecture#Security_Extensions_.28TrustZone.29), LPAE | Unknown | No |
| [ARMv8-A](https://en.wikipedia.org/wiki/ARMv8-A) | 64/32 | ARMv8-A[[3]](https://en.wikipedia.org/wiki/Comparison_of_instruction_set_architectures#cite_note-3) | 2011[[4]](https://en.wikipedia.org/wiki/Comparison_of_instruction_set_architectures#cite_note-4) | 3 | Register Register | RISC | 31 | Fixed (32-bit). In ARMv7 compatibility mode: Thumb: Fixed (16-bit), Thumb-2: Variable (16- and 32-bit), A64 | Condition code | Bi | None (all extensions of ARMv7 are non-optional) | Unknown | No |
| [AVR](https://en.wikipedia.org/wiki/Atmel_AVR_instruction_set) | 8 |  | 1997 | 2 | Register Register | RISC | 32 (16 on "reduced architecture") | Variable (mostly 16-bit, four instructions are 32-bit) | Condition register, skip conditioned on an I/O or general purpose register bit, compare and skip | Little |  | Unknown | Unknown |
| [AVR32](https://en.wikipedia.org/wiki/AVR32) | 32 | Rev 2 | 2006 | 2–3 |  | RISC | 15 | Variable[[5]](https://en.wikipedia.org/wiki/Comparison_of_instruction_set_architectures#cite_note-5) |  | Big | [Java Virtual Machine](https://en.wikipedia.org/wiki/Java_Virtual_Machine) | Unknown | Unknown |
| [Blackfin](https://en.wikipedia.org/wiki/Blackfin) | 32 |  | 2000 |  |  | RISC[[6]](https://en.wikipedia.org/wiki/Comparison_of_instruction_set_architectures#cite_note-6) | 8 |  |  | Little[[7]](https://en.wikipedia.org/wiki/Comparison_of_instruction_set_architectures#cite_note-7) |  | Unknown | Unknown |
| [CDC Cyber](https://en.wikipedia.org/wiki/CDC_Cyber) | 60 |  | 1970s | 3 | Register Memory | [RISC](https://en.wikipedia.org/wiki/Reduced_instruction_set_computer) | 24 (8 18-bit address registers, 8 18-bit index registers, 8 60-bit operand registers) | Variable (15, 30, and 60-bit) | Compare and branch | n/a[[8]](https://en.wikipedia.org/wiki/Comparison_of_instruction_set_architectures#cite_note-8) | Compare/Move Unit, additional Peripheral Processing Units | No | No |
| [Crusoe](https://en.wikipedia.org/wiki/Transmeta_Crusoe) (native VLIW) | 32[[9]](https://en.wikipedia.org/wiki/Comparison_of_instruction_set_architectures#cite_note-crusoe-arch-9) |  | 2000 | 1 | Register Register[[9]](https://en.wikipedia.org/wiki/Comparison_of_instruction_set_architectures#cite_note-crusoe-arch-9) | VLIW[[9]](https://en.wikipedia.org/wiki/Comparison_of_instruction_set_architectures#cite_note-crusoe-arch-9)[[10]](https://en.wikipedia.org/wiki/Comparison_of_instruction_set_architectures#cite_note-technology-behind-crusoe-10) | * 1 in native push stack mode * 6 in x86 emulation + 8 in x87/mmx mode + 50 in rename status * 12 integer + 48 shadow + 4 debug in native VLIW mode   [[9]](https://en.wikipedia.org/wiki/Comparison_of_instruction_set_architectures#cite_note-crusoe-arch-9)[[10]](https://en.wikipedia.org/wiki/Comparison_of_instruction_set_architectures#cite_note-technology-behind-crusoe-10) | Variable (64- or 128-bit)[[10]](https://en.wikipedia.org/wiki/Comparison_of_instruction_set_architectures#cite_note-technology-behind-crusoe-10) | Condition code[[9]](https://en.wikipedia.org/wiki/Comparison_of_instruction_set_architectures#cite_note-crusoe-arch-9) | Little |  |  |  |
| [DLX](https://en.wikipedia.org/wiki/DLX) | 32 |  | 1990 | 3 |  | RISC | 32 | Fixed (32-bit) |  | Big |  | Unknown | Unknown |
| [eSi-RISC](https://en.wikipedia.org/wiki/ESi-RISC) | 16/32 |  | 2009 | 3 | Register Register | RISC | 8–72 | Variable (16- or 32-bit) | Compare and branch and condition register | Bi | User-defined instructions | No | No |
| [Itanium](https://en.wikipedia.org/wiki/Itanium) (IA-64) | 64 |  | 2001 |  | Register Register | [EPIC](https://en.wikipedia.org/wiki/Explicitly_parallel_instruction_computing) | 128 |  | Condition register | Bi (selectable) | Intel Virtualization Technology | No | No |
| [M32R](https://en.wikipedia.org/wiki/M32R) | 32 |  | 1997 |  |  | RISC | 16 | Fixed (16- or 32-bit) |  | Bi |  | Unknown | Unknown |
| [Mico32](https://en.wikipedia.org/wiki/LatticeMico32) | 32 |  | 2006 | 3 | Register Register | RISC | 32[[11]](https://en.wikipedia.org/wiki/Comparison_of_instruction_set_architectures#cite_note-11) | Fixed (32-bit) | Compare and branch | Big | User-defined instructions | Yes[[12]](https://en.wikipedia.org/wiki/Comparison_of_instruction_set_architectures#cite_note-12) | Yes |
| [MIPS](https://en.wikipedia.org/wiki/MIPS_architecture) | 64 (32→64) | 5 | 1981 | 1–3 | Register Register | RISC | 4–32 (including "zero") | Fixed (32-bit) | Condition register | Bi | [MDMX](https://en.wikipedia.org/wiki/MDMX), [MIPS-3D](https://en.wikipedia.org/wiki/MIPS-3D) | Unknown | No |
| [MMIX](https://en.wikipedia.org/wiki/MMIX) | 64 |  | 1999 | 3 | Register Register | RISC | 256 | Fixed (32-bit) |  | Big |  | Yes | Yes |
| [NS320xx](https://en.wikipedia.org/wiki/NS320xx) | 32 |  | 1982 | 5 | Memory Memory | CISC | 8 | Variable Huffman coded, up to 23 bytes long | Condition code | Little | BitBlt instructions | Unknown | Unknown |
| [OpenRISC](https://en.wikipedia.org/wiki/OpenRISC) | 32, 64 |  | 2010 | 3 | Register Register | RISC | 16 or 32 | Fixed |  |  |  | Yes | Yes |
| [PA-RISC](https://en.wikipedia.org/wiki/PA-RISC) (HP/PA) | 64 (32→64) | 2.0 | 1986 | 3 | Register Register | RISC | 32 | Fixed (32-bit) | Compare and branch | Big → Bi | [Multimedia Acceleration eXtensions](https://en.wikipedia.org/wiki/Multimedia_Acceleration_eXtensions) (MAX), MAX-2 | No | Unknown |
| [PDP-11](https://en.wikipedia.org/wiki/PDP-11) | 16 |  | 1970–1990 | 3 | Memory Memory | [CISC](https://en.wikipedia.org/wiki/Complex_instruction_set_computing) | 8 (includes stack pointer, though any register can act as stack pointer) | Fixed (16) | Condition code | Little | Floating Point, Commercial Instruction Set | No | No |
| [PowerPC](https://en.wikipedia.org/wiki/PowerPC) | 32/64 (32→64) | 2.07[[13]](https://en.wikipedia.org/wiki/Comparison_of_instruction_set_architectures#cite_note-13) | 1991 | 3 | Register Register | RISC | 32 | Fixed (32-bit), Variable | Condition code | Big/Bi | [AltiVec](https://en.wikipedia.org/wiki/AltiVec), APU, [VSX](https://en.wikipedia.org/wiki/AltiVec#VSX), [Cell](https://en.wikipedia.org/wiki/Cell_%28microprocessor%29) | Yes | No |
| [RISC-V](https://en.wikipedia.org/wiki/RISC-V) | 32, 64, 128 |  | 2010 |  | Register Register | RISC | 32 (including "zero") | Variable | Compare and branch | Little |  | Yes | Yes |
| [RX](https://en.wikipedia.org/wiki/RX_microcontroller_family) | 64/32/16 |  | 2000 | 3 | Memory Memory | CISC | 4 integer + 4 address | Variable | Compare and branch | Little |  | Unknown | No |
| [S+core](https://en.wikipedia.org/wiki/S%2Bcore) | 16/32 |  | 2005 |  |  | RISC |  |  |  | Little |  | Unknown | Unknown |
| [SPARC](https://en.wikipedia.org/wiki/SPARC) | 64 (32→64) | OSA2015[[14]](https://en.wikipedia.org/wiki/Comparison_of_instruction_set_architectures#cite_note-14) | 1985 | 3 | Register Register | RISC | 32 (including "zero") | Fixed (32-bit) | Condition code | Big → Bi | [VIS](https://en.wikipedia.org/wiki/Visual_Instruction_Set) 1.0, 2.0, 3.0, 4.0 | Yes | Yes[[15]](https://en.wikipedia.org/wiki/Comparison_of_instruction_set_architectures#cite_note-15) |
| [SuperH](https://en.wikipedia.org/wiki/SuperH) (SH) | 32 |  | 1990s | 2 | Register Register / Register Memory | RISC | 16 | Fixed (16- or 32-bit), Variable | Condition code (single bit) | Bi |  | Unknown | Unknown |
| [System/360](https://en.wikipedia.org/wiki/IBM_System/360_architecture) / [System/370](https://en.wikipedia.org/wiki/System/370) / [z/Architecture](https://en.wikipedia.org/wiki/Z/Architecture) | 64 (32→64) |  | 1964 | 2 (most) 3 (FMA, distinct-operand facility) | Register Memory / Memory Memory / Register Register | CISC | 16 | Variable | Condition code | Big |  | Unknown | Unknown |
| [Transputer](https://en.wikipedia.org/wiki/Transputer) | 32 (4→64) |  | 1987 | 1 | [Stack machine](https://en.wikipedia.org/wiki/Stack_machine) | MISC | 0 | Variable (8 ~ 120 bytes) | Compare and branch | Little |  |  |  |
| [VAX](https://en.wikipedia.org/wiki/VAX) | 32 |  | 1977 | 6 | Memory Memory | CISC | 16 | Variable | Compare and branch | Little | VAX Vector Architecture | Unknown | Unknown |
| [Z80](https://en.wikipedia.org/wiki/Z80) | 8 |  | 1976 | 2 | Register Memory | CISC | 17 | Variable (8 to 32 bits) | Condition register | Little |  |  |  |
| **Architecture** | **Bits** | **Version** | **Introduced** | **Max # operands** | **Type** | **Design** | **Registers (excluding FP/vector)** | **Instruction encoding** | **Branch evaluation** | **Endianness** | **Extensions** | **Open** | **Royalty free** |

1. Instruction set untuk prosesor intel.

| **Instruction** | **Meaning** | **Notes** | **Opcode** |
| --- | --- | --- | --- |
| [AAA](https://en.wikipedia.org/wiki/Intel_BCD_opcodes) | ASCII adjust AL after addition | used with unpacked [binary coded decimal](https://en.wikipedia.org/wiki/Binary_coded_decimal) | 0x37 |
| AAD | ASCII adjust AX before division | 8086/8088 datasheet documents only base 10 version of the AAD instruction ([opcode](https://en.wikipedia.org/wiki/Opcode" \o "Opcode) 0xD5 0x0A), but any other base will work. Later Intel's documentation has the generic form too. NEC V20 and V30 (and possibly other NEC V-series CPUs) always use base 10, and ignore the argument, causing a number of incompatibilities | 0xD5 |
| AAM | ASCII adjust AX after multiplication | Only base 10 version (Operand is 0xA) is documented, see notes for AAD | 0xD4 |
| AAS | ASCII adjust AL after subtraction |  | 0x3f |
| ADC | Add with carry | destination := destination + source + [carry\_flag](https://en.wikipedia.org/wiki/Carry_flag) | 0x10…0x15, 0x80/2…0x83/2 |
| ADD | Add | (1) r/m += r/imm; (2) r += m/imm; | 0x00…0x05, 0x80/0…0x83/0 |
| AND | [Logical AND](https://en.wikipedia.org/wiki/Logical_conjunction) | (1) r/m &= r/imm; (2) r &= m/imm; | 0x20…0x25, 0x80/4…0x83/4 |
| CALL | Call procedure | push eip; eip points to the instruction directly after the call | 0x9A, 0xE8, 0xFF/2, 0xFF/3 |
| CBW | Convert byte to word |  | 0x98 |
| CLC | Clear [carry flag](https://en.wikipedia.org/wiki/Carry_flag) | CF = 0; | 0xF8 |
| CLD | Clear [direction flag](https://en.wikipedia.org/wiki/Direction_flag) | DF = 0; | 0xFC |
| [CLI](https://en.wikipedia.org/wiki/CLI_%28x86_instruction%29) | Clear [interrupt flag](https://en.wikipedia.org/wiki/IF_%28x86_flag%29) | IF = 0; | 0xFA |
| CMC | Complement carry flag |  | 0xF5 |
| CMP | Compare operands |  | 0x38…0x3D, 0x80/7…0x83/7 |
| CMPSB | Compare bytes in memory |  | 0xA6 |
| CMPSW | Compare words |  | 0xA7 |
| CWD | Convert word to doubleword |  | 0x99 |
| [DAA](https://en.wikipedia.org/wiki/Intel_BCD_opcodes) | Decimal adjust AL after addition | (used with packed [binary coded decimal](https://en.wikipedia.org/wiki/Binary_coded_decimal)) | 0x27 |
| [DAS](https://en.wikipedia.org/wiki/Intel_BCD_opcodes) | Decimal adjust AL after subtraction |  | 0x2F |
| DEC | Decrement by 1 |  | 0x48, 0xFE/1, 0xFF/1 |
| DIV | Unsigned divide | DX:AX = DX:AX / r/m; resulting DX == remainder | 0xF6/6, 0xF7/6 |
| ESC | Used with [floating-point unit](https://en.wikipedia.org/wiki/Floating-point_unit) |  |  |
| [HLT](https://en.wikipedia.org/wiki/HLT_%28x86_instruction%29) | Enter halt state |  | 0xF4 |
| IDIV | Signed divide | DX:AX = DX:AX / r/m; resulting DX == remainder | 0xF6/7, 0xF7/7 |
| IMUL | Signed multiply | (1) DX:AX = AX \* r/m; (2) AX = AL \* r/m | 0x69, 0x6B, 0xF6/5, 0xF7/5, 0x0FAF |
| IN | Input from port | (1) AL = port[imm]; (2) AL = port[DX]; (3) AX = port[DX]; | 0xE4, 0xE5, 0xEC, 0xED |
| INC | Increment by 1 |  | 0x40, 0xFE/0, 0xFF/0 |
| [INT](https://en.wikipedia.org/wiki/INT_%28x86_instruction%29) | Call to [interrupt](https://en.wikipedia.org/wiki/Interrupt) |  | 0xCD |
| INTO | Call to interrupt if overflow |  | 0xCE |
| IRET | Return from interrupt |  | 0xCF |
| Jcc | [Jump if condition](https://en.wikipedia.org/wiki/Branch_%28computer_science%29) | (JA, JAE, JB, JBE, JC, JE, JG, JGE, JL, JLE, JNA, JNAE, JNB, JNBE, JNC, JNE, JNG, JNGE, JNL, JNLE, JNO, JNP, JNS, JNZ, JO, JP, JPE, JPO, JS, JZ) | 0x70…0x7F, 0xE3, 0x0F83, 0x0F87 |
| JCXZ | Jump if CX is zero |  | 0xE3 |
| [JMP](https://en.wikipedia.org/wiki/JMP_%28x86_instruction%29) | Jump |  | 0xE9…0xEB, 0xFF/4, 0xFF/5 |
| LAHF | Load FLAGS into AH register |  | 0x9F |
| LDS | Load pointer using DS |  | 0xC5 |
| LEA | [Load Effective Address](https://en.wikipedia.org/wiki/Load_Effective_Address) |  | 0x8D |
| LES | Load ES with pointer |  | 0xC4 |
| LOCK | Assert BUS LOCK# signal | (for multiprocessing) | 0xF0 |
| LODSB | Load string byte | if (DF==0) AL = \*SI++; else AL = \*SI--; | 0xAC |
| LODSW | Load string word | if (DF==0) AX = \*SI++; else AX = \*SI--; | 0xAD |
| LOOP/LOOPx | Loop control | (LOOPE, LOOPNE, LOOPNZ, LOOPZ) if (x && --CX) goto lbl; | 0xE0..0xE2 |
| MOV | Move | copies data from one location to another, (1) r/m = r; (2) r = r/m; |  |
| MOVSB | Move byte from string to string | if (DF==0)  \*(byte\*)DI++ = \*(byte\*)SI++;  else  \*(byte\*)DI-- = \*(byte\*)SI--; | 0xA4 |
| MOVSW | Move word from string to string | if (DF==0)  \*(word\*)DI++ = \*(word\*)SI++;  else  \*(word\*)DI-- = \*(word\*)SI--; | 0xA5 |
| MUL | Unsigned multiply | (1) DX:AX = AX \* r/m; (2) AX = AL \* r/m; |  |
| NEG | Two's complement negation | r/m \*= -1; |  |
| [NOP](https://en.wikipedia.org/wiki/NOP) | No operation | opcode equivalent to XCHG EAX, EAX, but it is never optimized to "nothing happens", always using a fixed number of processor ticks | 0x90 |
| NOT | Negate the operand, [logical NOT](https://en.wikipedia.org/wiki/Bitwise_operation#NOT) | r/m ^= -1; |  |
| OR | [Logical OR](https://en.wikipedia.org/wiki/Logical_disjunction) | (1) r/m |= r/imm; (2) r |= m/imm; |  |
| OUT | Output to port | (1) port[imm] = AL; (2) port[DX] = AL; (3) port[DX] = AX; |  |
| POP | Pop data from [stack](https://en.wikipedia.org/wiki/Stack_%28data_structure%29) | r/m = \*SP++; POP CS (opcode 0x0F) works only on 8086/8088. Later CPUs use 0x0F as a prefix for newer instructions. |  |
| POPF | Pop [FLAGS register](https://en.wikipedia.org/wiki/FLAGS_register_%28computing%29) from stack | FLAGS = \*SP++; | 0x9D |
| PUSH | Push data onto stack | \*--SP = r/m; |  |
| PUSHF | Push FLAGS onto stack | \*--SP = FLAGS; | 0x9C |
| RCL | Rotate left (with carry) |  |  |
| RCR | Rotate right (with carry) |  |  |
| REPxx | Repeat MOVS/STOS/CMPS/LODS/SCAS | (REP, REPE, REPNE, REPNZ, REPZ) |  |
| RET | Return from procedure | Not a real instruction. The assembler will translate these to a RETN or a RETF depending on the memory model of the target system. |  |
| RETN | Return from near procedure |  |  |
| RETF | Return from far procedure |  |  |
| ROL | Rotate left |  |  |
| ROR | Rotate right |  |  |
| SAHF | Store AH into FLAGS |  | 0x9E |
| SAL | [Shift Arithmetically](https://en.wikipedia.org/wiki/Arithmetic_shift) left (signed shift left) | (1) r/m <<= 1; (2) r/m <<= CL; |  |
| SAR | Shift Arithmetically right (signed shift right) | (1) (signed) r/m >>= 1; (2) (signed) r/m >>= CL; |  |
| SBB | Subtraction with borrow | alternative 1-byte encoding of SBB AL, AL is available via [undocumented](https://en.wikipedia.org/wiki/X86_instruction_listings#Undocumented_instructions) SALC instruction |  |
| SCASB | Compare byte string |  | 0xAE |
| SCASW | Compare word string |  | 0xAF |
| SHL | [Shift](https://en.wikipedia.org/wiki/Logical_shift) left (unsigned shift left) |  |  |
| SHR | Shift right (unsigned shift right) |  |  |
| STC | Set carry flag | CF = 1; | 0xF9 |
| STD | Set direction flag | DF = 1; | 0xFD |
| [STI](https://en.wikipedia.org/wiki/STI_%28x86_instruction%29) | Set interrupt flag | IF = 1; | 0xFB |
| STOSB | Store byte in string | if (DF==0) \*ES:DI++ = AL; else \*ES:DI-- = AL; | 0xAA |
| STOSW | Store word in string | if (DF==0) \*ES:DI++ = AX; else \*ES:DI-- = AX; | 0xAB |
| SUB | Subtraction | (1) r/m -= r/imm; (2) r -= m/imm; |  |
| [TEST](https://en.wikipedia.org/wiki/TEST_%28x86_instruction%29) | Logical compare (AND) | (1) r/m & r/imm; (2) r & m/imm; |  |
| WAIT | Wait until not busy | Waits until BUSY# pin is inactive (used with [floating-point unit](https://en.wikipedia.org/wiki/Floating-point_unit)) | 0x9B |
| XCHG | Exchange data | r :=: r/m; A [spinlock](https://en.wikipedia.org/wiki/Spinlock) typically uses xchg as an [atomic operation](https://en.wikipedia.org/wiki/Atomic_operation). ([coma bug](https://en.wikipedia.org/wiki/Coma_bug)). |  |
| XLAT | Table look-up translation | behaves like MOV AL, [BX+AL] | 0xD7 |
| XOR | [Exclusive OR](https://en.wikipedia.org/wiki/Exclusive_or) | (1) r/m ^= r/imm; (2) r ^= m/imm; |  |