Tugas Praktikum Sistem Operasi Modul 1

Nama : Arya Mukti A'raafi Zha Putra

NIM : L200180151

Kelas : D

1. **- Apa yang dimaksud dengan kode 'ASCII'?**

   ASCII merupakan kepanjangan dari (American Standard Code for Information Interchange), dan pengertian dari ASCII sendiri adalah suatu standar internasional dalam kode huruf dan simbol seperti Hex dan Unicode tetapi ASCII lebih bersifat universal, contohnya 124 adalah untuk karakter "|". Ia selalu digunakan oleh komputer dan alat komunikasi lain untuk menunjukkan teks.

   - Buatlah tabel kode ASCII lengkap cukup kode ASCII yang standar tidak perlu extended.

Dec	Hex	Name	Char	Ctrl-char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
0	0	Null	NUL	CTRL-@	32	20	Space	64	40	@	96	60	`
1	1	Start of heading	SOH	CTRL-A	33	21	!	65	41	A	97	61	a
2	2	Start of text	STX	CTRL-B	34	22	"	66	42	B	98	62	b
3	3	End of text	ETX	CTRL-C	35	23	#	67	43	C	99	63	c
4	4	End of xmit	EOT	CTRL-D	36	24	$	68	44	D	100	64	d
5	5	Enquiry	ENQ	CTRL-E	37	25	%	69	45	E	101	65	e
6	6	Acknowledge	ACK	CTRL-F	38	26	&	70	46	F	102	66	f
7	7	Bell	BEL	CTRL-G	39	27	'	71	47	G	103	67	g
8	8	Backspace	BS	CTRL-H	40	28	(	72	48	H	104	68	h
9	9	Horizontal tab	HT	CTRL-I	41	29	)	73	49	I	105	69	i
10	0A	Line feed	LF	CTRL-J	42	2A	*	74	4A	J	106	6A	j
11	0B	Vertical tab	VT	CTRL-K	43	2B	+	75	4B	K	107	6B	k
12	0C	Form feed	FF	CTRL-L	44	2C	,	76	4C	L	108	6C	l
13	0D	Carriage feed	CR	CTRL-M	45	2D	-	77	4D	M	109	6D	m
14	0E	Shift out	SO	CTRL-N	46	2E	.	78	4E	N	110	6E	n
15	0F	Shift in	SI	CTRL-O	47	2F	/	79	4F	O	111	6F	o
16	10	Data line escape	DLE	CTRL-P	48	30	0	80	50	P	112	70	p
17	11	Device control 1	DC1	CTRL-Q	49	31	1	81	51	Q	113	71	q
18	12	Device control 2	DC2	CTRL-R	50	32	2	82	52	R	114	72	r
19	13	Device control 3	DC3	CTRL-S	51	33	3	83	53	S	115	73	s
20	14	Device control 4	DC4	CTRL-T	52	34	4	84	54	T	116	74	t
21	15	Neg acknowledge	NAK	CTRL-U	53	35	5	85	55	U	117	75	u
22	16	Synchronous idle	SYN	CTRL-V	54	36	6	86	56	V	118	76	v
23	17	End of xmit block	ETB	CTRL-W	55	37	7	87	57	W	119	77	w
24	18	Cancel	CAN	CTRL-X	56	38	8	88	58	X	120	78	x
25	19	End of medium	EM	CTRL-Y	57	39	9	89	59	Y	121	79	y
26	1A	Substitute	SUB	CTRL-Z	58	3A	:	90	5A	Z	122	7A	z
27	1B	Escape	ESC	CTRL-[	59	3B	;	91	5B	[	123	7B	{
28	1C	File separator	FS	CTRL-\	60	3C	<	92	5C	\	124	7C	|
29	1D	Group separator	GS	CTRL-]	61	3D	=	93	5D	]	125	7D	}
30	1E	Record separator	RS	CTRL-^	62	3E	>	94	5E	^	126	7E	~
31	1F	Unit separator	US	CTRL-_	63	3F	?	95	5F	_	127	7F	DEL

   - **Tuliskan kode ASCII dalam format angka desimal, binary dan hexadesimal serta karakter dan simbol yang dikodekan**

- Desimal: 8 → Hexadesimal: 08 → Binary: 00001000 → Karakter ASCII: BS
- Desimal: 64 → Hexadesimal: 40 → Binary: 01000000 → Symbol ASCII: @

- Desimal: 165 → Hexadesimal: A5 → Binary: 10100101 → Symbol ASCII: ¥
- Desimal: 13 → Hexadesimal: 0D → Binary: 00001101 → Karakter ASCII: CR

2. Carilah daftar perintah bahasa assembly untuk mesin intel keluarga x86

Original 8086/8088 instruction set			
Instruction	**Meaning**	**Notes**	**Opcode**
AAA	ASCII adjust AL after addition	used with unpacked binary coded decimal	0x37
AAD	ASCII adjust AX before division	8086/8088 datasheet documents only base 10 version of the AAD instruction (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`	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	(1) `r/m &= r/imm;` (2) `r &= m/imm;`	0x20…0x25, 0x80/4…0x83/4

Original 8086/8088 instruction set			
Instruction	**Meaning**	**Notes**	**Opcode**
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	`CF = 0;`	0xF8
CLD	Clear direction flag	`DF = 0;`	0xFC
CLI	Clear interrupt flag	`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	Decimal adjust AL after addition	(used with packed binary coded decimal)	0x27

Original 8086/8088 instruction set			
Instruction	**Meaning**	**Notes**	**Opcode**
DAS	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		
HLT	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	Call to interrupt		0xCD

Original 8086/8088 instruction set			
Instruction	**Meaning**	**Notes**	**Opcode**
INTO	Call to interrupt if overflow		0xCE
IRET	Return from interrupt		0xCF
Jcc	Jump if condition	(`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	Jump		0xE9...0xEB, 0xFF/4, 0xFF/5
LAHF	Load FLAGS into AH register		0x9F
LDS	Load pointer using DS		0xC5
LEA	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

Original 8086/8088 instruction set			
Instruction	**Meaning**	**Notes**	**Opcode**
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	No operation	opcode equivalent to `XCHG EAX, EAX`	0x90

Original 8086/8088 instruction set			
Instruction	**Meaning**	**Notes**	**Opcode**
NOT	Negate the operand, logical NOT	`r/m ^= -1;`	
OR	Logical OR	(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	`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 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.	
RETF	Return from far procedure		
RETN	Return from near procedure		
ROL	Rotate left		
ROR	Rotate right		
SAHF	Store AH into FLAGS		0x9E
SAL	Shift Arithmetically 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 SALC instruction	

Original 8086/8088 instruction set			
Instruction	**Meaning**	**Notes**	**Opcode**
SCASB	Compare byte string		0xAE
SCASW	Compare word string		0xAF
SHL	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	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	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)	0x9B
XCHG	Exchange data	`r :=: r/m;` A spinlock typically uses xchg as an atomic operation. (coma bug).	
XLAT	Table look-up translation	behaves like `MOV AL, [BX+AL]`	0xD7
XOR	Exclusive OR	(1) `r/m ^= r/imm;` (2) `r ^= m/imm;`	

Added in specific processors

Added with 80186/80188

Instruction	Meaning	Notes
BOUND	Check array index against bounds	raises software interrupt 5 if test fails
ENTER	Enter stack frame	Modifies stack for entry to procedure for high level language. Takes two operands: the amount of storage to be allocated on the stack and the nesting level of the procedure.

INS	Input from port to string	equivalent to
		`IN (E)AX, DX`
		`MOV ES:[(E)DI], (E)AX`
		`; adjust (E)DI according to operand size and DF`

LEAVE	Leave stack frame	Releases the local stack storage created by the previous ENTER instruction.

OUTS	Output string to port	equivalent to
		`MOV (E)AX, DS:[(E)SI]`
		`OUT DX, (E)AX`
		`; adjust (E)SI according to operand size and DF`

POPA	Pop all general purpose registers from stack	equivalent to
		`POP DI`
		`POP SI`
		`POP BP`
		`POP AX ;no POP SP here, only ADD SP,2`
		`POP BX`

		POP DX POP CX POP AX
PUSHA	Push all general purpose registers onto stack	equivalent to PUSH AX PUSH CX PUSH DX PUSH BX PUSH SP ; *The value stored is the initial SP value* PUSH BP PUSH SI PUSH DI
PUSH immediate	Push an immediate byte/word value onto the stack	equivalent to PUSH 12h PUSH 1200h
IMUL immediate	Signed multiplication of immediate byte/word value	equivalent to IMUL BX,12h IMUL DX,1200h IMUL CX, DX, 12h IMUL BX, SI, 1200h IMUL DI, word ptr [BX+SI], 12h IMUL SI, word ptr [BP-4], 1200h
SHL/SHR/SAL/SAR/ROL/ROR/RCL/RCR immediate	Rotate/shift bits with an immediate value greater than 1	equivalent to ROL AX,3 SHR BL,3

Added with 80286

Instruction	Meaning	Notes
ARPL	Adjust RPL field of selector	

CLTS	Clear task-switched flag in register CR0	
LAR	Load access rights byte	
LGDT	Load global descriptor table	
LIDT	Load interrupt descriptor table	
LLDT	Load local descriptor table	
LMSW	Load machine status word	
LOADALL	Load all CPU registers, including internal ones such as GDT	Undocumented, 80286 and 80386 only
LSL	Load segment limit	
LTR	Load task register	
SGDT	Store global descriptor table	
SIDT	Store interrupt descriptor table	
SLDT	Store local descriptor table	
SMSW	Store machine status word	
STR	Store task register	
VERR	Verify a segment for reading	
VERW	Verify a segment for writing	

Added with 80386

Instruction	Meaning	Notes
BSF	Bit scan forward	
BSR	Bit scan reverse	
BT	Bit test	
BTC	Bit test and complement	
BTR	Bit test and reset	
BTS	Bit test and set	
CDQ	Convert double-word to quad-word	Sign-extends EAX into EDX, forming the quad-word EDX:EAX. Since (I)DIV uses EDX:EAX as its input, CDQ must be called after setting EAX if EDX is not manually initialized (as in 64/32 division) before (I)DIV.
CMPSD	Compare string double-word	Compares ES:[(E)DI] with DS:[(E)SI] and increments or decrements both (E)DI and (E)SI, depending on DF; can be prefixed with REP
CWDE	Convert word to double-word	Unlike CWD, CWDE sign-extends AX to EAX instead of AX to DX:AX
IBTS	Insert Bit String	discontinued with B1 step of 80386
INSD	Input from port to string double-word	
IRETx	Interrupt return; D suffix means 32-bit return, F suffix means do	Use IRETD rather than IRET in 32-bit situations

	not generate epilogue code (i.e. LEAVE instruction)	
JECXZ	Jump if ECX is zero	
LFS, LGS	Load far pointer	
LSS	Load stack segment	
LODSD	Load string double-word	`EAX = *ES:EDI±±;` (**±±** depends on DF, ES cannot be overridden); can be prefixed with REP
LOOPW, LOOP*cc*W	Loop, conditional loop	Same as LOOP, LOOP*cc* for earlier processors
LOOPD, LOOPccD	Loop while equal	`if (`*cc*` && --ECX) goto lbl;`, *cc* = **Z**(ero), **E**(qual), **N**on**Z**ero, **N**(on)**E**(qual)
MOV to/from CR/DR/TR	Move to/from special registers	CR=control registers, DR=debug registers, TR=test registers (up to 80486)
MOVSD	Move string double-word	`*(dword*)ES:EDI±± = (dword*)ESI±±;` (**±±** depends on DF); can be prefixed with REP
MOVSX	Move with sign-extension	`(long)r = (signed char) r/m;` and similar
MOVZX	Move with zero-extension	`(long)r = (unsigned char) r/m;` and similar
OUTSD	Output to port from string double-word	`port[DX] = *(long*)ESI±±;` (**±±** depends on DF)

POPAD	Pop all double-word (32-bit) registers from stack	Does not pop register ESP off of stack
POPFD	Pop data into EFLAGS register	
PUSHAD	Push all double-word (32-bit) registers onto stack	
PUSHFD	Push EFLAGS register onto stack	
SCASD	Scan string data double-word	Compares ES:[(E)DI] with EAX and increments or decrements (E)DI, depending on DF; can be prefixed with REP
SETcc	Set byte to one on condition, zero otherwise	(`SETA, SETAE, SETB, SETBE, SETC, SETE, SETG, SETGE, SETL, SETLE, SETNA, SETNAE, SETNB, SETNBE, SETNC, SETNE, SETNG, SETNGE, SETNL, SETNLE, SETNO, SETNP, SETNS, SETNZ, SETO, SETP, SETPE, SETPO, SETS, SETZ`)
SHLD	Shift left double-word	
SHRD	Shift right double-word	`r1 = r1>>CL | r2<<(32-CL);` Instead of CL, immediate 1 can be used
STOSD	Store string double-word	`*ES:EDI±± = EAX;` (±± depends on DF, ES cannot be overridden); can be prefixed with REP
XBTS	Extract Bit String	discontinued with B1 step of 80386

Added with 80486

Instruction	Meaning	Notes
BSWAP	Byte Swap	`r = r<<24 | r<<8&0x00FF0000 | r>>8&0x0000FF00 | r>>24;` Only works for 32 bit registers
CMPXCHG	atomic CoMPare and eXCHanGe	See Compare-and-swap / on later 80386 as undocumented opcode available
INVD	Invalidate Internal Caches	Flush internal caches
INVLPG	Invalidate TLB Entry	Invalidate TLB Entry for page that contains data specified
WBINVD	Write Back and Invalidate Cache	Writes back all modified cache lines in the processor's internal cache to main memory and invalidates the internal caches.
XADD	eXchange and ADD	Exchanges the first operand with the second operand, then loads the sum of the two values into the destination operand.

Added with Pentium

Instruction	Meaning	Notes
CPUID	CPU IDentification	Returns data regarding processor identification and features, and returns data to the EAX, EBX, ECX, and EDX registers. Instruction functions specified by the EAX register.[1] This was also added to later 80486 processors
CMPXCHG8B	CoMPare and eXCHanGe 8 bytes	Compare EDX:EAX with m64. If equal, set ZF and load ECX:EBX into m64. Else, clear ZF and load m64 into EDX:EAX.
RDMSR	ReaD from Model-specific register	Load MSR specified by ECX into EDX:EAX

RDTSC	ReaD Time Stamp Counter	Returns the number of processor ticks since the processor being "ONLINE" (since the last power on of system)
WRMSR	WRite to Model-Specific Register	Write the value in EDX:EAX to MSR specified by ECX
RSM[2]	Resume from System Management Mode	This was introduced by the i386SL and later and is also in the i486SL and later. Resumes from System Management Mode (SMM)

Added with Pentium MMX

Instruction	Meaning	Notes
RDPMC	Read the PMC [Performance Monitoring Counter]	Specified in the ECX register into registers EDX:EAX

Also MMX registers and MMX support instructions were added. They are usable for both integer and floating point operations, see below.

Added with AMD K6

Instruction	Meaning	Notes
SYSCALL		functionally equivalent to SYSENTER
SYSRET		functionally equivalent to SYSEXIT

AMD changed the CPUID detection bit for this feature from the K6-II on.

Added with Pentium Pro

Instruction	Meaning	Notes
CMOVcc	Conditional move	*(CMOVA, CMOVAE, CMOVB, CMOVBE, CMOVC, CMOVE, CMOVG, CMOVGE, CMOVL, CMOVLE, CMOVNA, CMOVNAE, CMOVNB, CMOVNBE, CMOVNC, CMOVNE, CMOVNG, CMOVNGE, CMOVNL, CMOVNLE, CMOVNO, CMOVNP, CMOVNS, CMOVNZ, CMOVO, CMOVP, CMOVPE, CMOVPO, CMOVS, CMOVZ)*

UD2	Undefined Instruction	Generates an invalid opcode. This instruction is provided for software testing to explicitly generate an invalid opcode. The opcode for this instruction is reserved for this purpose.

Added with Pentium II

Instruction	Meaning	Notes
SYSENTER	SYStem call ENTER	Sometimes called the Fast System Call instruction, this instruction was intended to increase the performance of operating system calls. Note that on the Pentium Pro, the CPUID instruction incorrectly reports these instructions as available.
SYSEXIT	SYStem call EXIT	

Added with SSE

Instruction	Opcode	Meaning	Notes
MASKMOVQ mm1, mm2	0F F7 /r	Masked Move of Quadword	Selectively write bytes from mm1 to memory location using the byte mask in mm2
MOVNTPS m128, xmm1	0F 2B /r	Move Aligned Four Packed Single-FP Non Temporal	Move packed single-precision floating-point values from xmm1 to m128, minimizing pollution in the cache hierarchy.
MOVNTQ m64, mm	0F E7 /r	Move Quadword Using Non-Temporal Hint	
NOP r/m16	0F 1F /0	Multi-byte no-operation instruction.	
NOP r/m32			
PREFETCHT0	0F 18 /1	Prefetch Data from Address	Prefetch into all cache levels

PREFETCHT1	0F 18 /2	Prefetch Data from Address	Prefetch into all cache levels EXCEPT[3][4] L1
PREFETCHT2	0F 18 /3	Prefetch Data from Address	Prefetch into all cache levels EXCEPT L1 and L2
PREFETCHNTA	0F 18 /0	Prefetch Data from Address	Prefetch to non-temporal cache structure, minimizing cache pollution.
SFENCE	0F AE F8	Store Fence	Processor hint to make sure all store operations that took place prior to the SFENCE call are globally visible

Added with SSE2

Instruction	Opcode	Meaning	Notes
CLFLUSH m8	0F AE /7	Cache Line Flush	Invalidates the cache line that contains the linear address specified with the source operand from all levels of the processor cache hierarchy
LFENCE	0F AE E8	Load Fence	Serializes load operations.
MFENCE	0F AE F0	Memory Fence	Performs a serializing operation on all load and store instructions that were issued prior the MFENCE instruction.
MOVNTI m32, r32	0F C3 /r	Move Doubleword Non-Temporal	Move doubleword from r32 to m32, minimizing pollution in the cache hierarchy.
PAUSE	F3 90	Spin Loop Hint	Provides a hint to the processor that the following code is a spin loop, for cacheability

Added with SSE3

Instruction	Meaning	Notes
`MONITOR EAX, ECX, EDX`	Setup Monitor Address	Sets up a linear address range to be monitored by hardware and activates the monitor.
`MWAIT EAX, ECX`	Monitor Wait	Processor hint to stop instruction execution and enter an implementation-dependent optimized state until occurrence of a class of events.

Added with SSE4.2

Instruction	Opcode	Meaning	Notes
CRC32 r32, r/m8	F2 0F 38 F0 /r	Accumulate CRC32	Computes CRC value using the CRC-32C (Castagnoli) polynomial 0x11EDC6F41 (normal form 0x1EDC6F41). This is the polynomial used in iSCSI. In contrast to the more popular one used in Ethernet, its parity is even, and it can thus detect any error with an odd number of changed bits.
CRC32 r32, r/m8	F2 REX 0F 38 F0 /r		
CRC32 r32, r/m16	F2 0F 38 F1 /r		
CRC32 r32, r/m32	F2 0F 38 F1 /r		
CRC32 r64, r/m8	F2 REX.W 0F 38 F0 /r		
CRC32 r64, r/m64	F2 REX.W 0F 38 F1 /r		
CRC32 r32, r/m8	F2 0F 38 F0 /r		

Added with x86-64

Instruction	Meaning	Notes
CDQE	Sign extend EAX into RAX	
CQO	Sign extend RAX into RDX:RAX	
CMPSQ	CoMPare String Quadword	
CMPXCHG16B	CoMPare and eXCHanGe 16 Bytes	
IRETQ	64-bit Return from Interrupt	
JRCXZ	Jump if RCX is zero	
LODSQ	LOaD String Quadword	
MOVSXD	MOV with Sign Extend 32-bit to 64-bit	
POPFQ	POP RFLAGS Register	
PUSHFQ	PUSH RFLAGS Register	
RDTSCP	ReaD Time Stamp Counter and Processor ID	
SCASQ	SCAn String Quadword	
STOSQ	STOre String Quadword	
SWAPGS	Exchange GS base with KernelGSBase MSR	

Added with AMD-V

Instruction	Meaning	Notes	Opcode
CLGI	Clear Global Interrupt Flag	Clears the GIF	0x0F 0x01 0xDD
INVLPGA	Invalidate TLB entry in a specified ASID	Invalidates the TLB mapping for the virtual page specified in RAX and the ASID specified in ECX.	0x0F 0x01 0xDF
MOV(CRn)	Move to or from control registers	Moves 32- or 64-bit contents to control register and vice versa.	0x0F 0x22 or 0x0F 0x20
MOV(DRn)	Move to or from debug registers	Moves 32- or 64-bit contents to control register and vice versa.	0x0F 0x21 or 0x0F 0x23
SKINIT	Secure Init and Jump with Attestation	Verifiable startup of trusted software based on secure hash comparison	0x0F 0x01 0xDE
STGI	Set Global Interrupt Flag	Sets the GIF.	0x0F 0x01 0xDC
VMLOAD	Load state From VMCB	Loads a subset of processor state from the VMCB specified by the physical address in the RAX register.	0x0F 0x01 0xDA
VMMCALL	Call VMM	Used exclusively to communicate with VMM	0x0F 0x01 0xD9
VMRUN	Run virtual machine	Performs a switch to the guest OS.	0x0F 0x01 0xD8
VMSAVE	Save state To VMCB	Saves additional guest state to VMCB.	0x0F 0x01 0xDB

Added with Intel VT-x

Instruction	Meaning	Notes	Opcode
VMPTRLD	Load Pointer to Virtual-Machine Control Structure	Loads the current VMCS pointer from memory.	0x0F 0xC7/6
VMPTRST	Store Pointer to Virtual-Machine Control Structure	Stores the current-VMCS pointer into a specified memory address. The operand of this instruction is always 64 bits and is always in memory.	0x0F 0xC7/7
VMCLEAR	Clear Virtual-Machine Control Structure	Writes any cached data to the VMCS	0x66 0x0F 0xC7/6
VMREAD	Read Field from Virtual-Machine Control Structure	Reads out a field in the VMCS	0x0F 0x78
VMWRITE	Write Field to Virtual-Machine Control Structure	Modifies a field in the VMCS	0x0F 0x79
VMCALL	Call to VM Monitor	Calls VM Monitor function from Guest System	0x0F 0x01 0xC1
VMLAUNCH	Launch Virtual Machine	Launch virtual machine managed by current VMCS	0x0F 0x01 0xC2
VMRESUME	Resume Virtual Machine	Resume virtual machine managed by current VMCS	0x0F 0x01 0xC3
VMXOFF	Leave VMX Operation	Stops hardware supported virualisation environment	0x0F 0x01 0xC4
VMXON	Enter VMX Operation	Enters hardware supported virualisation environment	0xF3 0x0F 0xC7/6