PRAKTIKUM SISTEM OPERASI

TUGAS MODUL 1

PENGENALAN SISTEM PENGEMBANGAN OS

DENGAN PC SIMULATOR "BOCHS"

DISUSUN:

NAMA : RAHMAD NUR SULISTYAWAN

NIM : L200190194

KELAS : F

INFORMATIKA

FAKULTAS KOMUNIKASI DAN INFORMATIKA

UNIVERSITAS MUHAMMADIYAH SURAKARTA

2020

TUGAS SISTEM OPERASI

1. Apa yang dimaksud dengan kode ‘ASCII’, buatlah tabel kode ASCII lengkap cukup kode ASCII yang standar tidak perlu extended, tuliskan kode ASCII dalam format angka desimal, binary, dan hexadesimal serta karakter dan simbol yang dikodekan.

Jawab:

ASCII yaitu singkatan dari American Standard Code for Information Interchange atau Kode Standar Amerika untuk Pertukaran Informasi adalah suatu standar pengkodean karakter untuk alat komunikasi. Kode ASCII mewakili teks dalam komputer, peralatan telekomunikasi, dan perangkat lainnya. Kebanyakan skema pengkodean karakter modern didasarkan pada ASCII, meskipun mereka mendukung banyak karakter tambahan.

DESIMAL	BINARY	HEXADESIMAL	SIMBOL / KARAKTER	KETERANGAN
000	00000000	000	NUL	Null char \ 0
001	00000001	001	SOH	Start of Header
002	00000010	002	STX	Strart of Text
003	00000011	003	ETX	End of Text
004	00000100	004	EOT	End of Transmission
005	00000101	005	ENG	Enquiry
006	00000110	006	ACK	Acknowledment
007	00000111	007	BEL	Bell \ a
008	00001000	008	BS	Backspace \ b
009	00001001	009	HT	Horizontal \ t
010	00001010	00A	LF	Line Feed \ n
011	00001011	00B	VT	Vertical Tab \ v
012	00001100	00C	FF	Form Feed \ f
013	00001101	00D	CR	Carriage Return \ r
014	00001110	00E	SO	Shift Out
015	00001111	00F	SI	Shift In
016	00010000	010	DLE	Data Link Escape
017	00010001	011	DC1	XON, Device Control 1
018	00010010	012	DC2	Device Control 2
019	00010011	013	DC3	XOFF, Device Control 3
020	00010100	014	DC4	Device Control 4
021	00010101	015	NAK	Negative Acknowledgement
022	00010110	016	SYN	Synchronous Idle

023	00010111	017	ETB	End of Trans. Block
024	00011000	018	CAN	Cancel
025	00011001	019	EM	End of Medium
026	00011010	01A	SUB	Subtitute
027	00011011	01B	ESC	Escape
028	00011100	01C	FS	File Separator
029	00011101	01D	GS	Group Separator
030	00011110	01E	RS	Request to Send Record Separator
031	00011111	01F	US	Unit Separator
032	00100000	020	SP	Space
033	00100001	021	!	Exclamation mark
034	00100010	022	“	Double quote
035	00100011	023	#	Number sign
036	00100100	024	$	Dollar sign
037	00100101	025	%	Percent
038	00100110	026	&	Ampersand
039	00100111	027	‘	Single quote
040	00101000	028	(	Left / Opening paranthesis
041	00101001	029	)	Right / Closing parenthesis
042	00101010	02A	*	Asterisk
043	00101011	02B	+	Plus
044	00101100	02C	,	Comma
045	00101101	02D	-	Minus or dash
046	00101110	02E	.	Dot
047	00101111	02F	/	Forward slash
048	00110000	030	0	
049	00110001	031	1	
050	00110010	032	2	
051	00110011	033	3	
052	00110100	034	4	
053	00110101	035	5	
054	00110110	036	6	
055	00110111	037	7	
056	00111000	038	8	
057	00111001	039	9	
058	00111010	03A	:	Colon

059	00111011	03B	;	Semi-colon
060	00111100	03C	<	Less than
061	00111101	03D	=	Equal sign
062	00111110	03E	>	Greater than
063	00111111	03F	?	Question mark
064	01000000	040	@	AT symbol
065	01000001	041	A	
066	01000010	042	B	
067	01000011	043	C	
068	01000100	044	D	
069	01000101	045	E	
070	01000110	046	F	
071	01000111	047	G	
072	01001000	048	H	
073	01001001	049	I	
074	01001010	04A	J	
075	01001011	04B	K	
076	01001100	04C	L	
077	01001101	04D	M	
078	01001110	04E	N	
079	01001111	04F	O	
080	01010000	050	P	
081	01010001	051	Q	
082	01010010	052	R	
083	01010011	053	S	
084	01010100	054	T	
085	01010101	055	U	
086	01010110	056	V	
087	01010111	057	W	
088	01011000	058	X	
089	01011001	059	Y	
090	01011010	05A	Z	
091	01011011	05B	[	Left / Opening bracket
092	01011100	05C	\	Back slash
093	01011101	05D	]	Right / Close bracket
094	01011110	05E	^	Caret / Circumflex
095	01011111	05F	_	Underscore

096	01100000	060	`	
097	01100001	061	a	
098	01100010	062	b	
099	01100011	063	c	
100	01100100	064	d	
101	01100101	065	e	
102	01100110	066	f	
103	01100111	067	g	
104	01101000	068	h	
105	01101001	069	i	
106	01101010	06A	j	
107	01101011	06B	k	
108	01101100	06C	l	
109	01101101	06D	m	
110	01101110	06E	n	
111	01101111	06F	o	
112	01110000	070	p	
113	01110001	071	q	
114	01110010	072	r	
115	01110011	073	s	
116	01110100	074	t	
117	01110101	075	u	
118	01110110	076	v	
119	01110111	077	w	
120	01111000	078	x	
121	01111001	079	y	
122	01111010	07A	z	
123	01111011	07B	{	Left / Opening brace
124	01111100	07C	|	Vertical bar
125	01111101	07D	}	Right / Closing brace
126	01111110	07E	~	Tilde
127	01111111	07F	DEL	Delete

2. Carilah daftar perintah bahasa assembly untuk mesin intel keluarga x86 lengkap (dari buku referensi atau dari internet). Daftar perintah ini dapat digunakan sebagai pedoman untuk memahami program *'boot.asm'* dan *'kernel.asm*

Jawab:

1. **ACALL (Absolute Call)**
   ACALL berfungsi untuk memanggil sub rutin program
2. A**DD (Add Immediate Data)**
   ADD berfungsi untuk menambah 8 bit data langsung ke dalam isi akumulator dan menyimpan hasilnya pada akumulator.
3. **ADDC (Add Carry Plus Immediate Data to Accumulator)**
   ADDC berfungsi untuk menambahkan isi carry flag (0 atau 1) ke dalam isi akumulator. Data langsung 8 bit ditambahkan ke akumulator.
4. **AJMP (Absolute Jump)**
   AJMP adalah perintah jump mutlak. Jump dalam 2 KB dimulai dari alamat yang mengikuti perintah AJMP. AJMP berfungsi untuk mentransfer kendali program ke lokasi dimana alamat dikalkulasi dengan cara yang sama dengan perintah ACALL. Konter program ditambahkan dua kali dimana perintah AJMP adalah perintah 2-byte. Konter program di-load dengan a10 – a0 11 bits, untuk membentuk alamat tujuan 16-bit.
5. **ANL (logical AND memori ke akumulator)**
   ANL berfunsi untuk mengAND-kan isi alamat data dengan isi akumulator.
6. **CJNE (Compare Indirect Address to Immediate Data)**
   CJNE berfungsi untuk membandingkan data langsung dengan lokasi memori yang dialamati oleh register R atau Akumulator A. apabila tidak sama maka instruksi akan menuju ke alamat kode. Format : CJNE R,#data,Alamat kode.
7. **CLR (Clear Accumulator)**
   CLR berfungsi untuk mereset data akumulator menjadi 00H.
   Format : CLR A
8. **CPL (Complement Accumulator)**
   CPL berfungsi untuk mengkomplemen isi akumulator.
9. **DA (Decimal Adjust Accumulator)**
   DA berfungsi untuk mengatur isi akumulator ke padanan BCD, steleah penambahan dua angka BCD.
10. **DEC (Decrement Indirect Address)**
    DEC berfungsi untuk mengurangi isi lokasi memori yang ditujukan oleh register R dengan 1, dan hasilnya disimpan pada lokasi tersebut.
11. **DIV (Divide Accumulator by B)**
    DIV berfungsi untuk membagi isi akumulator dengan isi register B. Akumulator berisi hasil bagi, register B berisi sisa pembagian.
12. **DJNZ (Decrement Register And Jump Id Not Zero)**
    DJNZ berfungsi untuk mengurangi nilai register dengan 1 dan jika hasilnya sudah 0 maka instruksi selanjutnya akan dieksekusi. Jika belum 0 akan menuju ke alamat kode.
13. **INC (Increment Indirect Address)**
    INC berfungsi untuk menambahkan isi memori dengan 1 dan menyimpannya pada alamat tersebut.

14. **JB (Jump if Bit is Set)**

    JB berfungsi untuk membaca data per satu bit, jika data tersebut adalah 1 maka akan menuju ke alamat kode dan jika 0 tidak akan menuju ke alamat kode.

15. **JBC (Jump if Bit Set and Clear Bit)**

    Bit JBC, berfungsi sebagai perintah rel menguji yang terspesifikasikan secara bit. Jika bit di-set, maka Jump dilakukan ke alamat relatif dan yang terspesifikasi secara bit di dalam perintah dibersihkan. Segmen program berikut menguji bit yang kurang signifikan (LSB: Least Significant Byte), dan jika diketemukan bahwa ia telah di-set, program melompat ke READ lokasi. JBC juga berfungsi membersihkan LSB dari akumulator.

16. **JC (Jump if Carry is Set)**

    Instruksi JC berfungsi untuk menguji isi carry flag. Jika berisi 1, eksekusi menuju ke alamat kode, jika berisi 0, instruksi selanjutnya yang akan dieksekusi.

17. **JMP (Jump to sum of Accumulator and Data Pointer)**

    Instruksi JMP berfungsi untuk memerintahkan loncat kesuato alamat kode tertentu.

    Format : JMP alamat kode.

18. **JNB (Jump if Bit is Not Set)**

    Instruksi JNB berfungsi untuk membaca data per satu bit, jika data tersebut adalah 0 maka akan menuju ke alamat kode dan jika 1 tidak akan menuju ke alamat kode.

    Format : JNB alamat bit,alamat kode.

19. **JNC (Jump if Carry Not Set)**

    JNC berfungsi untuk menguji bit Carry, dan jika tidak di-set, maka sebuah lompatan akan dilakukan ke alamat relatif yang telah ditentukan.

20. **JNZ (Jump if Accumulator Not Zero)**

    JNZ adalah mnemonik untuk instruksi jump if not zero (lompat jika tidak nol). Dalam hal ini suatu lompatan akan terjadi bilamana bendera nol dalam keadaan “clear”, dan tidak akan terjadi lompatan bilamana bendera nol tersebut dalam keadaan set. Andaikan bahwa JNZ 7800H disimpan pada lokasi 2100H. Jika Z=0, instruksi berikutnya akan berasal dari lokasi 7800H: dan bilamana Z=1, program akan turun ke instruksi urutan berikutnya pada lokasi 2101H.

21. **JZ (Jump if Accumulator is Zero)**

    JZ berfungsi untuk menguji konten-konten akumulator. Jika bukan nol, maka lompatan dilakukan ke alamat relatif yang ditentukan dalam perintah.

22. **LCALL (Long Call)**

    LCALL berfungsi untuk memungkinkan panggilan ke subrutin yang berlokasi dimanapun dalam memori program 64K. Operasi LCALL berjalan seperti berikut:

    - Menambahkan ke dalam konter program sebanyak 3, karena perintahnya adalah perintah 3-byte.
    - Menambahkan penunjuk stack sebanyak 1.
    - Menyimpan byte yang lebih rendah dari konter program ke dalam stack.
    - Menambahkan penunjuk stack.
    - Menyimpan byte yang lebih tinggi dari program ke dalam stack.
    - Me-load konter program dengan alamat tujuan 16-bit.

23. LJMP (Long Jump)

Long Jump befungsi untuk memungkinkan lompatan tak bersyarat kemana saja dalam lingkup ruang memori program 64K. LCALL adalah perintah 3-byte. Alamat tujuan 16-bit ditentukan secara langsung dalam perintah tersebut. Alamat tujuan ini di-load ke dalam konter program oleh perintah LJMP.

24. MOV (Move From Memory)

MOV berfungsi untuk memindahkan isi akumulator/register atau data dari nilai luar atau alamat lain.

25. MOVC (Move From Codec Memory)

Instruksi MOVC berfungsi untuk mengisi accumulator dengan byte kode atau konstanta dari program memory. Alamat byte tersebut adalah hasil penjumlahan unsigned 8 bit pada accumulator dan 16 bit register basis yang dapat berupa data pointer atau program counter. Instruksi ini tidak mempengaruhi flag apapun juga.

26. MOVX (Move Accumulator to External Memory Addressed by Data Pointer)

MOVX berfungsi untuk memindahkan isi akumulator ke memori data eksternal yang alamatnya ditunjukkan oleh isi data pointer.

27. MUL (Multiply)

MUL AB berfungsi untuk mengalikan unsigned 8 bit integer pada accumulator dan register B. Byte rendah (low order) dari hasil perkalian akan disimpan dalam accumulator sedangkan byte tinggi (high order) akan disimpan dalam register B. Jika hasil perkalian lebih besar dari 255 (0FFh), overflow flag akan bernilai ‘1’. Jika hasil perkalian lebih kecil atau sama dengan 255, overflow flag akan bernilai ‘0’. Carry flag akan selalu dikosongkan.

28. NOP (No Operation)

Fungsi NOP adalah eksekusi program akan dilanjutkan ke instruksi berikutnya. Selain PC, instruksi ini tidak mempengaruhi register atau flag apapun juga.

29. ORL (Logical OR Immediate Data to Accumulator)

Instruksi ORL berfungsi sebagai instruksi Gerbang logika OR yang akan menjumlahkan Accumulator terhadap nilai yang ditentukan.

Format : ORL A,#data.

30. POP (Pop Stack to Memory)

Instruksi POP berfungsi untuk menempatkan byte yang ditunjukkan oleh stack pointer ke suatu alamat data.

31. PUSH (Push Memory onto Stack)

Instruksi PUSH berfungsi untuk menaikkan stack pointer kemudian menyimpan isinya ke suatu alamat data pada lokasi yang ditunuk oleh stack pointer.

32. RET (Return from subroutine)

Intruksi RET berfungsi untuk kembali dari suatu subrutin program ke alamat terakhir subrutin tersebut di panggil.

33. RETI (Return From Interrupt)

RETI berfungsi untuk mengambil nilai byte tinggi dan rendah dari PC dari stack dan mengembalikan kondisi logika interrupt agar dapat menerima interrupt lain dengan prioritas yang sama dengan prioritas interrupt yang baru saja diproses. Stack pointer akan dikurangi dengan 2. Instruksi ini tidak mempengaruhi flag apapun juga. Nilai PSW tidak akan dikembalikan secara otomatis ke kondisi sebelum interrupt. Eksekusi program akan dilanjutkan pada alamat yang diambil tersebut.

Umumnya alamat tersebut adalah alamat setelah lokasi dimana terjadi interrupt. Jika interrupt dengan prioritas sama atau lebih rendah tertunda saat RETI dieksekusi, maka satu instruksi lagi akan dieksekusi sebelum interrupt yang tertunda tersebut diproses.

34. RL (Rotate Accumulator Left)

Instruksi RL berfungsi untuk memutar setiap bit dalam akumulator satu posisi ke kiri.

35. RLC (Rotate Left through Carry)

Fungsi : Memutar (Rotate) Accumulator ke Kiri (Left) Melalui Carry Flag. Kedelapan bit accumulator dan carry flag akan diputar satu bit ke kiri secara bersama-sama. Bit 7 akan dirotasi ke carry flag, nilai carry flag akan berpindah ke posisi bit 0. Instruksi ini tidak mempengaruhi flag lain.

36. RR (Rotate Right)

Fungsi : Memutar (Rotate) Accumulator ke Kanan (Right). Kedelapan bit accumulator akan diputar satu bit ke kanan. Bit 0 akan dirotasi ke posisi bit 7. Instruksi ini tidak mempengaruhi flag apapun juga.

37. RRC (Rotate Right through Carry)

Fungsi : Memutar (Rotate) Accumulator ke Kanan (Right) Melalui Carry Flag. Kedelapan bit accumulator dan carry flag akan diputar satu bit ke kanan secara bersama-sama. Bit 0 akan dirotasi ke carry flag, nilai carry flag akan berpindah ke posisi bit 7. Instruksi ini tidak mempengaruhi flag lain.

38. SETB (set Carry flag)

Instruksi SETB berfungsi untuk menset carry flag.

39. SJMP (Short Jump)

Sebuah Short Jump berfungsi untuk mentransfer kendali ke alamat tujuan dalam 127 bytes yang mengikuti dan 128 yang mengawali perintah SJMP. Alamat tujuannya ditentukan sebagai sebuat alamat relative 8-bit. Ini adalah Jump tidak bersyarat. Perintah SJMP menambahkan konter program sebanyak 2 dan menambahkan alamat relatif ke dalamnya untuk mendapatkan alamat tujuan. Alamat relatif tersebut ditentukan dalam perintah sebagai ‘SJMP rel’.

40. SUBB (Subtract With Borrow)

Fungsi : Pengurangan (Subtract) dengan Peminjaman (Borrow). SUBB mengurangi variabel yang tertera pada operand kedua dan carry flag sekaligus dari accumulator dan menyimpan hasilnya pada accumulator. SUBB akan memberi nilai ‘1’ pada carry flag jika peminjaman ke bit 7 dibutuhkan dan mengosongkan C jika tidak dibutuhkan peminjaman. Jika C bernilai ‘1’ sebelum mengeksekusi SUBB, hal ini menandakan bahwa terjadi peminjaman pada proses pengurangan sebelumnya, sehingga carry flag dan source byte akan dikurangkan dari accumulator secara bersama-sama. AC akan bernilai ‘1’ jika peminjaman ke bit 3 dibutuhkan dan mengosongkan AC jika tidak dibutuhkan peminjaman. OV akan bernilai ‘1’ jika ada peminjaman ke bit 6 namun tidak ke bit 7 atau ada peminjaman ke bit 7 namun tidak ke bit 6. Saat mengurangi signed integer, OV menandakan adanya angka negative sebagai hasil dari pengurangan angka negatif dari angka positif atau adanya angka positif sebagai hasil dari pengurangan angka positif dari angka negative. Addressing mode yang dapat digunakan adalah: register, direct, register indirect, atau immediate data.

41. **SWAP (Swap Nibbles)**

    Fungsi : Menukar (Swap) Upper Nibble dan Lower Nibble dalam Accumulator. SWAP A akan menukar nibble (4 bit) tinggi dan nibble rendah dalam accumulator. Operasi ini dapat dianggap sebagai rotasi 4 bit dengan RR atau RL. Instruksi ini tidak mempengaruhi flag apapun juga.

42. **XCH (Exchange Bytes)**

    Fungsi : Menukar (Exchange) Accumulator dengan Variabel Byte. XCH akan mengisi accumulator dengan variabel yang tertera pada operand kedua dan pada saat yang sama juga akan mengisikan nilai accumulator ke dalam variabel tersebut. Addressing mode yang dapat digunakan adalah: register, direct, atau register indirect.

43. **XCHD (Exchange Digits)**

    Fungsi : Menukar (Exchange) Digit. XCHD menukar nibble rendah dari accumulator, yang umumnya mewakili angka heksadesimal atau BCD, dengan nibble rendah dari internal data memory yang diakses secara indirect. Nibble tinggi kedua register tidak akan terpengaruh. Instruksi ini tidak mempengaruhi flag apapun juga.

44. **XRL (Exclusive OR Logic)**

    Fungsi : Logika Exclusive OR untuk Variabel Byte XRL akan melakukan operasi bitwise logika exclusive OR antara kedua variabel yang dinyatakan. Hasilnya akan disimpan pada destination byte. Instruksi ini tidak mempengaruhi flag apapun juga. Kedua operand mampu menggunakan enam kombinasi addressing mode. Saat destination byte adalah accumulator, source byte dapat berupa register, direct, register indirect, atau immediate data. Saat destination byte berupa direct address, source byte dapat berupa accumulator atau immediate data.

Sumber : https://davidmaulanamakruf.wordpress.com/2013/09/25/perintah-dasar-assembler-dan-fungsinya/