LAPORAN PRAKTIKUM
PENGENALAN SISTEM PENGEMBANGAN OS
PRAKTIKUM SISTEM OPERASI

Oleh:

RAIHAN GHASSANI

L200190123

PROGRAM STUDI INFORMATIKA

FAKULTAS KOMUNIKASI DAN INFORMATIKA

UNIVERSITAS MUHAMMADIYAH SURAKARTA

1. Apa yang dimaksud dengan kode ‘ASCII’, buatlah table 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.

   ASCII (American Standard Code for Information Interchage) merupakan standar pengkodean karakter dan symbol seperti Unicode dan Hex tetapi ASCII lebih bersifat universal untuk alat komunikasi. Kode ASCII mewakili teks dalam computer, peralatan telekomunikasi dan perangkat lainnya.

   Kode Standar Amerika untuk Pertukaran Informasi atau ASCII (American Standard Code for Information Interchange) merupakan suatu standar internasional dalam kode huruf dan simbol sepertiHex dan Unicode tetapi ASCII lebih bersifat universal, contohnya 124 adalah untuk karakter”|” ASCII adalah singkatan dari American Standard Code for Information Interchange. Sesuai dengan namanya, ASCII digunakan untuk pertukaran informasi dan komunikasi data. ASCII merupakan kode angka yang mewakili sebuah karakter. ASCII digunakan karena komputer hanya mengerti angka-angka. Secara sederhana ASCII merupakan kode standar yang digunakan dalam pertukaran informasi pada Komputer.

   Table Kode ASCII :

Desimal	Heksa Desimal	Karakter	**Biner**
0	0000	NUL	0000 0000
1	0001	SOH	0000 0001
2	0002	STX	0000 0010
3	0003	ETX	0000 0011
4	0004	EOT	0000 0100
5	0005	ENQ	0000 0101
6	0006	ACK	0000 0110
7	0007	BEL	0000 0111
8	0008	BS	0000 1000
9	0009	HT	0000 1001
10	000A	LF	0000 1010
11	000B	VT	0000 1011

12	000C	FF	0000 1100
13	000D	CR	0000 1101
14	000E	SO	0000 1110
15	000F	SI	0000 1111
16	0010	DLE	0001 0000
17	0011	DC1	0001 0001
18	0012	DC2	0001 0010
19	0013	DC3	0001 0011
20	0014	DC4	0001 0100

21	0015	NAK	0001 0101
22	0016	SYN	0001 0110
23	0017	ETB	0001 0111
24	0018	CAN	0001 1000
25	0019	EM	0001 1001
26	001A	SUB	0001 1010
27	001B	ESC	0001 1011
28	001C	FS	0001 1100
29	001D	GS	0001 1101
30	001E	RS	0001 1110
31	001F	US	0001 1111
32	0020	spasi	0010 0000
33	0021	!	0010 0001
34	0022	“	0010 0010
35	0023	#	0010 0011
36	0024	$	0010 0100
37	0025	%	0010 0101
38	0026	&	0010 0110
39	0027	‘	0010 0111
40	0028	(	0010 1000
41	0029	)	0010 1001
42	002A	*	0010 1010
43	002B	+	0010 1011
44	002C	,	0010 1100
45	002D	-	0010 1101
46	002E	.	0010 1110
47	002F	/	0010 1111
48	0030	0	0011 0000
49	0031	1	0011 0001
50	0032	2	0011 0010
51	0033	3	0011 0011
52	0034	4	0011 0100
53	0035	5	0011 0101

54	0036	6	0011 0110
55	0037	7	0011 0111
56	0038	8	0011 1000
57	0039	9	0011 1001
58	003A	:	0011 1010
59	003B	;	0011 1011
60	003C	<	0011 1100
61	003D	=	0011 1101
62	003E	>	0011 1110
63	003F	?	0011 1111
64	0040	@	0100 0000
65	0041	A	0100 0001

66	0042	B	0100 0010
67	0043	C	0100 0011
68	0044	D	0100 0100
69	0045	E	0100 0101
70	0046	F	0100 0110
71	0047	G	0100 0111
72	0048	H	0100 1000
73	0049	I	0100 1001
74	004A	J	0100 1010
75	004B	K	0100 1011
76	004C	L	0100 1100
77	004D	M	0100 1101
78	004E	N	0100 1110
79	004F	O	0100 1111
80	0050	P	0101 0000
81	0051	Q	0101 0001
82	0052	R	0101 0010
83	0053	S	0101 0011
84	0054	T	0101 0100
85	0055	U	0101 0101
86	0056	V	0101 0110
87	0057	W	0101 0111
88	0058	X	0101 1000
89	0059	Y	0101 1001
90	005A	Z	0101 1010
91	005B	[	0101 1011
92	005C	/	0101 1100
93	005D	]	0101 1101
94	005E	^	0101 1110
95	005F	_	0101 1111
96	0060	`	0110 0000
97	0061	a	0110 0001
98	0062	b	0110 0010

99	0063	c	0110 0011
100	0064	d	0110 0100
101	0065	e	0110 0101
102	0066	f	0110 0110
103	0067	g	0110 0111
104	0068	h	0110 1000
105	0069	i	0110 1001
106	006A	j	0110 1010
107	006B	k	0110 1011
108	006C	l	0110 1100
109	006D	m	0110 1101
110	006E	n	0110 1110

111	006F	o	0110 1111
112	0070	p	0111 0000
113	0071	q	0111 0001
114	0072	r	0111 0010
115	0073	s	0111 0011
116	0074	t	0111 0100
117	0075	u	0111 0101
118	0076	v	0111 0110
119	0077	w	0111 0111
120	0078	x	0111 1000
121	0079	y	0111 1001
122	007A	z	0111 1010
123	007B	{	0111 1011
124	007C	|	0111 1100
125	007D	}	0111 1101
126	007E	~	0111 1110
127	007F	DEL	0111 1111

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

1. ACALL (Absolute Call)

ACALL berfungsi untuk memanggil sub rutin program

2. ADD (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.