PRAKTIKUM SISTEM OPERASI

MODUL 1

Disusun Oleh:

NAMA : SURYA SUMIRAT

NIM : L200190217

KELAS : F

INFORMATIKA

FAKULTAS KOMUNIKASI DAN INFORMATIKA

UNIVERSITAS MUHAMMADIYAH SURAKARTA

2020 / 2021

Tugas

1. Apa yang dimaksud dengan kode ASCII, buatlah tabel kode ASCII lengkap cukup kode ASCII yang standar tidak perlu extend, tuliskan kode ASCII dalam format angka decimal, binary dan hexadecimal serta karakter dan symbol yang dikodekan.

   (**ASCII**) singkatan dari ***American Standard Code for Information Interchange*** atau **Kode Standar Amerika untuk Pertukaran Informasi** adalah standar pengkodean karakter untuk alat komunikasi. Kode ASCII mewakili teks dalam komputer, peralatan telekomunikasi, dan perangkat lainnya.

Nilai ANSI ASCII (Desimal)	Nilai Unicode (Heksa Desimal)	Karakter	Keterangan	Binary
13	00D	CR	Enter/CR	0001101
16	0010	DLE	Escape	0010000
8	0008	BS	Backspace	0001000
32	0020	spasi	Spasi	0100000
48	0030	0	Angka nol	0110000
49	0031	1	Angka satu	0110001
50	0032	2	Angka dua	0110010
51	0033	3	Angka tiga	0110011
52	0034	4	Angka empat	0110100
53	0035	5	Angka lima	0110101
54	0036	6	Angka enam	0110110
55	0037	7	Angka tujuh	0110111
56	0038	8	Angka delapan	0111000
57	0039	9	Angka sebilan	0111001
127	007F	DEL	Delete	1111111
66	0042	B	Huruf latin B kapital	1000010
67	0043	C	Huruf latin C kapital	1000011
68	0044	D	Huruf latin D kapital	1000100
69	0045	E	Huruf latin E kapital	1000101
70	0046	F	Huruf latin F kapital	1000110
71	0047	G	Huruf latin G kapital	1000111
72	0048	H	Huruf latin H kapital	1001000
73	0049	I	Huruf latin I kapital	1001001

74	004A	J	Huruf latin J kapital	1001010

75	004B	K	Huruf latin K kapital	1001011
76	004C	L	Huruf latin L kapital	1001100
77	004D	M	Huruf latin Mkapital	1001101
78	004E	N	Huruf latin N kapital	1001110
79	004F	O	Huruf latin O kapital	1001111
80	0050	P	Huruf latin P kapital	1010000
81	0051	Q	Huruf latin Q kapital	1010001
82	0052	R	Huruf latin R kapital	1010010
83	0053	S	Huruf latin S kapital	1010011
84	0054	T	Huruf latin T kapital	1010100
85	0055	U	Huruf latin U kapital	1010101
86	0056	V	Huruf latin V kapital	1010110
87	0057	W	Huruf latin Wkapital	1010111
88	0058	X	Huruf latin X kapital	1011000
89	0059	Y	Huruf latin Y kapital	1011001
90	005A	Z	Huruf latin Z kapital	1011010
97	0061	a	Huruf latin a kecil	1100001
98	0062	b	Huruf latin b kecil	1100010
99	0063	c	Huruf latin c kecil	1100011
100	0064	d	Huruf latin d kecil	1100100
101	0065	e	Huruf latin e kecil	1100101
102	0066	f	Huruf latin f kecil	1100110
103	0067	g	Huruf latin g kecil	1100111
104	0068	h	Huruf latin h kecil	1101000

105	0069	i	Huruf latin i kecil	1101001
106	006A	j	Huruf latin j kecil	1101010
107	006B	k	Huruf latin k kecil	1101011
108	006C	l	Huruf latin l kecil	1101100
109	006D	m	Huruf latin m kecil	1101101
110	006E	n	Huruf latin n kecil	1101110
111	006F	o	Huruf latin o kecil	1101111
112	0070	p	Huruf latin p kecil	1110000
113	0071	q	Huruf latin q kecil	1110001
114	0072	r	Huruf latin r kecil	1110010
115	0073	s	Huruf latin s kecil	1110011
116	0074	t	Huruf latin t kecil	1110100
117	0075	u	Huruf latin u kecil	1110101
118	0076	v	Huruf latin v kecil	1110110
119	0077	w	Huruf latin w kecil	1110111
120	0078	x	Huruf latin x kecil	1111000
121	0079	y	Huruf latin y kecil	1111001
122	007A	z	Huruf latin z kecil	1111010

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

    1. ANL (logical AND memori ke akumulator)

    ANL berfunsi untuk mengAND-kan isi alamat data dengan isi akumulator.

    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. ACALL (Absolute Call)

ACALL berfungsi untuk memanggil sub rutin program

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.

8. 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.

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.