PRAKTIKUM SISTEM OPERASI

MODUL 1: PENGENALAN DAN PENGEMBANGAN OS DENGAN PC SIMULATOR "BOCHS"

Disusun Oleh :

DWIKI REZA NOVA ALVIANTO

L200210235

KELAS E

PROGRAM STUDI TEKNIK INFORMATIKA

FAKULTAS KOMUNIKASI DAN INFORMATIKA

UNIVERSITAS MUHAMMADIYAH SURAKARTA

TAHUN 2022/2023

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**.

   Kode ASCII adalah singkatan dari American Standard Code for Information Interchangeatau Kode Standar Amerika untuk Pertukaran Informasi, Kode ASCII mewakili teks dalam komputer, peralatan telekomunikasi, dan perangkat lainnya. Kebanyakan skema pengkodean karakter modern didasarkan pada ASCII, meskipunmereka mendukung banyak karakter tambahan.

KARAKTER	DESIMAL	BINARY	HEXADESIMAL
0	48	00110000	0030
1	49	00110001	0031
2	50	00110010	0032
3	51	00110011	0033
4	52	00110100	0034
5	53	00110101	0035
6	54	00110110	0036
7	55	00110111	0037
8	56	00111000	0038
9	57	00111001	0039
A	65	01000001	0041
B	66	01000010	0042
C	67	01000011	0043
D	68	01000100	0044
E	69	01000101	0045
F	70	01000110	0046
G	71	01000111	0047
H	72	01001000	0048
I	73	01001001	0049
J	74	01001010	004A
K	75	01001011	004B
L	76	01001100	004C
M	77	01001101	004D
N	78	01001110	004E
O	79	01001111	004F
P	80	01010000	0050
Q	81	01010001	0051
R	82	01010010	0052
S	83	01010011	0053
T	84	01010100	0054
U	85	01010101	0055
V	86	01010110	0056
W	87	01010111	0057
X	88	01011000	0058

Y	89	01011001	0059
Z	90	01011010	005A
a	97	01100001	0061
b	98	01100010	0062
c	99	01100011	0063
d	100	01100100	0064
e	101	01100101	0065
f	102	01100110	0066
g	103	01100111	0067
h	104	01101000	0068
i	105	01101001	0069
j	106	01101010	006A
k	107	01101011	006B
l	108	01101100	006C
m	109	01101101	006D
n	110	01101110	006E
o	111	01101111	006F
p	112	01110000	0070
q	113	01110001	0071
r	114	01110010	0072
s	115	01110011	0073
t	116	01110100	0074
u	117	01110101	0075
v	118	01110110	0076
w	119	01110111	0077
x	120	01111000	0078
y	121	01111001	0079
z	122	01111010	007A

2. **Carilah daftar perintah bahasa assembly untuk mesin intel keluarga x86 lengkap (dari buku referensi atau internet). Daftar perintah ini dapat digunakan sebagai pedoman untukmemahami 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 berfungsi 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 Ratau 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, setelah 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 adalah1 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 SignificantByte), 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 untukmemerintahkan loncat kesuato alamat kode tertentu. Format : JMP alamat kode.

18. **JNB** (Jump if Bit is Not Set) Instruksi JNB berfungsi untuk membaca data persatu bit, jika data tersebut adalah 0 maka akan menuju ke alamat kode dan jika 1 tidak akan menuju ke alamatkode. Format : JNB alamat bit,alamat kode.

19. **JNC** (Jump if Carry Not Set) JNC berfungsi untuk menguji bit Carry, dan jikatidak 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 memindah 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 dataeksternal 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 darisuatu subrutin program ke alamat terakhir subrutin tersebut di panggil.

33. **RETI** (Return From Interrupt) RETI berfungsi untuk mengambil nilai byte tinggidan 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 memutarsetiap bit dalamakumulator 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 secarabersama-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 bitaccumulator 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 keKanan (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 keposisi bit 7. Instruksi ini tidak mempengaruhi flaglain.
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 tujuandalam 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' padacarry 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 terjadipeminjaman pada proses pengurangan sebelumnya, sehingga carry flag dan source byte akan dikurangkan dari accumulator secara bersama-sama. AC akan bernilai '1' jika peminjaman ke bit3 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 modeyang 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 nibblerendah 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 operan 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.