# LAPORAN PRAKTIKUM

# PEAKTIKUM SISTEM OPERASI

# KEGIATAN 1

**Putra Agung Khoiron Rafi'i**
**L200190145**

**PRODI INFORMATIKA**
**FAKULTAS KOMUNIKASI DAN INFORMATIKA**

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.

   **ASCII** (American Standard Code for Information Interchange) merupakan Kode Standar Amerika untuk Pertukaran Informasi atau sebuah standar internasional dalam pengkodean huruf dan simbol seperti Unicode dan Hex tetapi ASCII lebih bersifat universal.

| | | | | |
|---|---|---|---|---|
| 0 | 00 | 00000000 | NUL | Null char |
| 1 | 01 | 00000001 | SOH | Start of Heading |
| 2 | 02 | 00000010 | STX | Start of Text |
| 3 | 03 | 00000011 | ETX | End of Text |
| 4 | 04 | 00000100 | EOT | End of Transmission |
| 5 | 05 | 00000101 | ENQ | Enquiry |
| 6 | 06 | 00000110 | ACK | Acknowledgment |
| 7 | 07 | 00000111 | BEL | Bell |
| 8 | 08 | 00001000 | BS | Back Space |
| 9 | 09 | 00001001 | HT | Horizontal Tab |
| 10 | 0A | 00001010 | LF | Line Feed |
| 11 | 0B | 00001011 | VT | Vertical Tab |
| 12 | 0C | 00001100 | FF | Form Feed |
| 13 | 0D | 00001101 | CR | Carriage Return |
| 14 | 0E | 00001110 | SO | Shift Out / X-On |
| 15 | 0F | 00001111 | SI | Shift In / X-Off |
| 16 | 10 | 00010000 | DLE | Data Line Escape |
| 17 | 11 | 00010001 | DC1 | Device Control 1 (oft. XON) |
| 18 | 12 | 00010010 | DC2 | Device Control 2 |
| 19 | 13 | 00010011 | DC3 | Device Control 3 (oft. XOFF) |
| 20 | 14 | 00010100 | DC4 | Device Control 4 |
| 21 | 15 | 00010101 | NAK | Negative Acknowledgement |
| 22 | 16 | 00010110 | SYN | Synchronous Idle |
| 23 | 17 | 00010111 | ETB | End of Transmit Block |
| 24 | 18 | 0001100 | CAN | Cancel |

| | | | | |
|---|---|---|---|---|
| | | 0 | | |
| 25 | 19 | 0001100<br>1 | EM | End of Medium |
| 26 | 1A | 0001101<br>0 | SUB | Substitute |
| 27 | 1B | 0001101<br>1 | ESC | Escape |
| 28 | 1C | 0001110<br>0 | FS | File Separator |
| 29 | 1D | 0001110<br>1 | GS | Group Separator |
| 30 | 1E | 0001111<br>0 | RS | Record Separator |
| 31 | 1F | 0001111<br>1 | US | Unit Separator |
| 32 | 20 | 0010000<br>0 | | Space |
| 33 | 21 | 0010000<br>1 | ! | Exclamation mark |
| 34 | 22 | 0010001<br>0 | " | Double quotes (or speech marks) |
| 35 | 23 | 0010001<br>1 | # | Number |
| 36 | 24 | 0010010<br>0 | $ | Dollar |
| 37 | 25 | 0010010<br>1 | % | Per cent sign |

| | | | | |
|---|---|---|---|---|
| 38 | 26 | 00100110 | & | Ampersand |
| 39 | 27 | 00100111 | ' | Single quote |
| 40 | 28 | 00101000 | ( | Open parenthesis (or open bracket) |
| 41 | 29 | 00101001 | ) | Close parenthesis (or close bracket) |
| 42 | 2A | 00101010 | * | Asterisk |
| 43 | 2B | 00101011 | + | Plus |
| 44 | 2C | 00101100 | , | Comma |
| 45 | 2D | 00101101 | - | Hyphen |
| 46 | 2E | 00101110 | . | Period, dot or full stop |
| 47 | 2F | 00101111 | / | Slash or divide |
| 48 | 30 | 00110000 | 0 | Zero |
| 49 | 31 | 00110001 | 1 | One |
| 50 | 32 | 00110010 | 2 | Two |
| 51 | 33 | 00110011 | 3 | Three |
| 52 | 34 | 00110100 | 4 | Four |
| 53 | 35 | 00110101 | 5 | Five |
| 54 | 36 | 00110110 | 6 | Six |
| 55 | 37 | 00110111 | 7 | Seven |
| 56 | 38 | 00111000 | 8 | Eight |
| 57 | 39 | 00111001 | 9 | Nine |
| 58 | 3A | 00111010 | : | Colon |
| 59 | 3B | 00111011 | ; | Semicolon |
| 60 | 3C | 00111100 | < | Less than (or open angled bracket) |
| 61 | 3D | 00111101 | = | Equals |
| 62 | 3E | 00111110 | > | Greater than (or close angled bracket) |
| 63 | 3F | 00111111 | ? | Question mark |
| 64 | 40 | 01000000 | @ | At symbol |
| 65 | 41 | 01000001 | A | Uppercase A |
| 66 | 42 | 01000010 | B | Uppercase B |
| 67 | 43 | 0100001 | C | Uppercase C |

| | | | | |
|---|---|---|---|---|
| | | 1 | | |
| 68 | 44 | 01000100 | D | Uppercase D |
| 69 | 45 | 01000101 | E | Uppercase E |
| 70 | 46 | 01000110 | F | Uppercase F |
| 71 | 47 | 01000111 | G | Uppercase G |
| 72 | 48 | 01001000 | H | Uppercase H |
| 73 | 49 | 01001001 | I | Uppercase I |
| 74 | 4A | 01001010 | J | Uppercase J |
| 75 | 4B | 01001011 | K | Uppercase K |
| 76 | 4C | 01001100 | L | Uppercase L |
| 77 | 4D | 01001101 | M | Uppercase M |
| 78 | 4E | 01001110 | N | Uppercase N |
| 79 | 4F | 01001111 | O | Uppercase O |
| 80 | 50 | 01010000 | P | Uppercase P |
| 81 | 51 | 01010001 | Q | Uppercase Q |
| 82 | 52 | 01010010 | R | Uppercase R |
| 83 | 53 | 01010011 | S | Uppercase S |
| 84 | 54 | 01010100 | T | Uppercase T |
| 85 | 55 | 01010101 | U | Uppercase U |

| | | | | |
|---|---|---|---|---|
| 86 | 56 | 01010110 | V | Uppercase V |
| 87 | 57 | 01010111 | W | Uppercase W |
| 88 | 58 | 01011000 | X | Uppercase X |
| 89 | 59 | 01011001 | Y | Uppercase Y |
| 90 | 5A | 01011010 | Z | Uppercase Z |
| 91 | 5B | 01011011 | [ | Opening bracket |
| 92 | 5C | 01011100 | \ | Backslash |
| 93 | 5D | 01011101 | ] | Closing bracket |
| 94 | 5E | 01011110 | ^ | Caret - circumflex |
| 95 | 5F | 01011111 | _ | Underscore |
| 96 | 60 | 01100000 | ` | Grave accent |
| 97 | 61 | 01100001 | a | Lowercase a |
| 98 | 62 | 01100010 | b | Lowercase b |
| 99 | 63 | 01100011 | c | Lowercase c |
| 100 | 64 | 01100100 | d | Lowercase d |
| 101 | 65 | 01100101 | e | Lowercase e |
| 102 | 66 | 01100110 | f | Lowercase f |
| 103 | 67 | 01100111 | g | Lowercase g |
| 104 | 68 | 01101000 | h | Lowercase h |
| 105 | 69 | 01101001 | i | Lowercase i |
| 106 | 6A | 01101010 | j | Lowercase j |
| 107 | 6B | 01101011 | k | Lowercase k |
| 108 | 6C | 01101100 | l | Lowercase l |
| 109 | 6D | 01101101 | m | Lowercase m |
| 110 | 6E | 01101110 | n | Lowercase n |
| 111 | 6F | 01101111 | o | Lowercase o |
| 112 | 70 | 01110000 | p | Lowercase p |
| 113 | 71 | 01110001 | q | Lowercase q |
| 114 | 72 | 01110010 | r | Lowercase r |
| 115 | 73 | 0111001 | s | Lowercase s |

| | | | | |
|---|---|---|---|---|
| | | 1 | | |
| 116 | 74 | 01110100 | t | Lowercase t |
| 117 | 75 | 01110101 | u | Lowercase u |
| 118 | 76 | 01110110 | v | Lowercase v |
| 119 | 77 | 01110111 | w | Lowercase w |
| 120 | 78 | 01111000 | x | Lowercase x |
| 121 | 79 | 01111001 | y | Lowercase y |
| 122 | 7A | 01111010 | z | Lowercase z |
| 123 | 7B | 01111011 | { | Opening brace |
| 124 | 7C | 01111100 | \| | Vertical bar |
| 125 | 7D | 01111101 | } | Closing brace |
| 126 | 7E | 01111110 | ~ | Equivalency sign - tilde |
| 127 | 7F | 01111111 | | Delete |

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