Modul 2

NIM : L200180132

Nama : Rohana Murniati Furshotun

Nama Asisten : Salsa Sasmita M

Nur Aini A

Tanggal Praktikum : 17 September 2019

Tugas

1. Apa yang dimaksud dengan kode 'ASCII'. Buatlah tabel kode ASCII lengkap cukup kode ASCII yang standar tidak perlu extended, tuliskan kode dan simbol yang dikodekan.
   Jawab :
   - Kode ASCII atau *American Standard Code for Information Interchange* adalah suatu standar internasional dalam kode huruf dan symbol seperti Hex dan Unicode tetapi ASCII lebih bersifat universal.
     Bersifat universal contohnya, kode 124 adalah untuk karakter "|", yangmana selalu digunakan oleh computer dan alat komunikasi lain untuk menunjukkan teks.
   - Fungsi kode ASCII adalah digunakan untuk mewakili karakter-karakter angka maupun huruf di dalam computer, contoh pada karakter 1, 2, 3, A, B dan seterusnya.
   - Standar kode ASCII menggunakan 7 buah bit, karakter yang tersedia meliputi karakter control, huruf (A-Z dan a-z), digit (0-9), dan sejumlah symbol seperti + dan *. Namu, ASCII dikembangkan dengan menggunakan 8 buah bit dengan tambahan sejumlah symbol Yunani dan karakter grafis.
   - Tabel Standar Kode ASCII

Karakter Kontrol ASCII			
DEC	HEX	BINER	SIMBOL
00	00h	0	NULL
01	01h	1	SOH
02	02h	10	STX
03	03h	11	ETX
04	04h	100	EOT
05	05h	101	ENQ

06	06h	110	ACK
07	07h	111	BEL
08	08h	1000	BS
09	09h	1001	HT
10	0Ah	1010	LF
11	0Bh	1011	VT
12	0Ch	1100	FF
13	0Dh	1101	CR
14	0Eh	1110	SO
15	0Fh	1111	SI
16	10h	10000	DLE
17	11h	10001	DC1
18	12h	10010	DC2
19	13h	10011	DC3
20	14h	10100	DC4
21	15h	10101	NAK
22	16h	10110	SYN
23	17h	10111	ETB
24	18h	11000	CAN
25	19h	11001	EM
26	1Ah	11010	SUB
27	1Bh	11011	ESC
28	1Ch	11100	FSS
29	1Dh	11101	GS
30	1Eh	11110	RS
31	1Fh	11111	US
127	7Fh	1111111	DEL

ASCII Printable Character							
DEC	HEX	BINER	SIMBOL	DEC	HEX	BINER	SIMBOL
32	20h	100000	SPACE	64	40h	1000000	@
33	21h	100001	!	65	41h	1000001	A
34	22h	100010	"	66	42h	1000010	B
35	23h	100011	#	67	43h	1000011	C
36	24h	100100	$	68	44h	1000100	D
37	25h	100101	%	69	45h	1000101	E
38	26h	100110	&	70	46h	1000110	F
39	27h	100111	‘	71	47h	1000111	G

40	28h	101000	(	72	48h	1001000	H
41	29h	101001	)	73	49h	1001001	I
42	2Ah	101010	*	74	4Ah	1001010	J
43	2Bh	101011	+	75	4Bh	1001011	K
44	2Ch	101100	,	76	4Ch	1001100	L
45	2Dh	101101	-	77	4Dh	1001101	M
46	2Eh	101110	.	78	4Eh	1001110	N
47	2Fh	101111	/	79	4Fh	1001111	O
48	30h	110000	0	80	50h	1010000	P
49	31h	110001	1	81	51h	1010001	Q
50	32h	110010	2	82	52h	1010010	R
51	33h	110011	3	83	53h	1010011	S
52	34h	110100	4	84	54h	1010100	T
53	35h	110101	5	85	55h	1010101	U
54	36h	110110	6	86	56h	1010110	V
55	37h	110111	7	87	57h	1010111	W
56	38h	111000	8	88	58h	1011000	X
57	39h	111001	9	89	59h	1011001	Y
58	3Ah	111010	:	90	5Ah	1011010	Z
59	3Bh	111011	;	91	5Bh	1011011	[
60	3Ch	111100	<	92	5Ch	1011100	\
61	3Dh	111101	=	93	5Dh	1011101	]
62	3Eh	111110	>	94	5Eh	1011110	^
63	3Fh	111111	?	95	5Fh	1011111	_

DEC	HEX	BINER	SIMBOL
96	60h	1100000	`
97	61h	1100001	a
98	62h	1100010	b
99	63h	1100011	c
100	64h	1100100	d
101	65h	1100101	e
102	66h	1100110	f
103	67h	1100111	g
104	68h	1101000	h
105	69h	1101001	i
106	6Ah	1101010	j
107	6Bh	1101011	k
108	6Ch	1101100	l
109	6Dh	1101101	m
110	6Eh	1101110	n
111	6Fh	1101111	o
112	70h	1110000	p
113	71h	1110001	q
114	72h	1110010	r
115	73h	1110011	s
116	74h	1110100	t
117	75h	1110101	u
118	76h	1110110	v
119	77h	1110111	w
120	78h	1111000	x
121	79h	1111001	y
122	7Ah	1111010	z
123	7Bh	1111011	{
124	7Ch	1111100	|
125	7Dh	1111101	}
126	7Eh	1111110	~

2. Carilah daftar perintah bahasa assembly untuk mesin intel keluarga x86 lengkap.
   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.

Lompat-lompat

JMP: jump. Lompat tanpa syarat. Lompat begitu saja. sintaks: JMP {label tujuan}
Lompat bersyarat sintaksnya sama dengan JMP, yaitu perintah jump diikuti label tujuan.

PERINTAH	ARTI	SYARAT	KASUS	KETERANGAN ("OP" = OPERAND)	MENGIKUTI CMP?
JA	jump if above	CF = 0 ∧ ZF = 0	unsigned	lompat bila op 1 > op 2	ya
JNBE	jump if not below or equal				
JB	jump if below	CF = 1 ∧ ZF = 0	unsigned	lompat bila op 1 < op 2	ya
JNAE	jump if not above or equal				
JAE	jump if above or equal	CF = 0 ∨ ZF = 1	unsigned	lompat bila op 1 ≥ op 2	ya
JNB	jump if not below				
JBE	jump if below or equal	CF = 1 ∨ ZF = 1	unsigned	lompat bila op 1 ≤ op 2	ya
JNA	jump if not above				
JG	jump if greater	OF = 0 ∧ ZF = 0	signed	lompat bila op 1 > op 2	ya
JNLE	jump if not less or equal				
JGE	jump if greater or equal	OF = 0 ∨ ZF = 1	signed	lompat bila op 1 ≥ op 2	ya
JNL	jump if not less than				
JL	jump if less than	OF = 1 ∧ ZF = 0	signed	lompat bila op 1 < op 2	ya
JNGE	jump if not				

	greater or equal				
JLE	jump if less or equal	OF = 1 ∨ ZF = 1	signed	lompat bila op 1 ≤ op 2	ya
JNG	jump if not greater				
JE	jump if equal	ZF = 1	keduanya	lompat bila op 1 = op 2	ya
JZ	jump if zero	ZF = 1	keduanya	lompat bila op 1 = op 2	ya
JNE	jump if not equal	ZF = 0	keduanya	lompat bila op 1 ≠ op 2	ya
JNZ	jump if not zero	ZF = 0	keduanya	lompat bila op 1 ≠ op 2	ya
JC	jump if carry	CF = 1	N/A	lompat bila carry flag = 1	tidak
JNC	jump if not carry	CF = 0	N/A	lompat bila carry flag = 0	tidak
JP	jump on parity	PF = 1	N/A	lompat bila parity flag = 1	tidak selalu
JPE	jump on parity even			lompat bila bilangan genap	
JNP	jump on not parity	PF = 0	N/A	lompat bila parity flag = 0	tidak selalu
JPO	jump on parity odd			lompat bila bilangan ganjil	
JO	jump if overflow	OF = 1	N/A	lompat bila overflow flag = 1	tidak
JNO	jump if not overflow	OF = 0	N/A	lompat bila overflow flag = 0	tidak
JS	jump if sign	SF = 1	N/A	lompat bila bilangan negatif	tidak
JCXZ	jump if CX is zero	CX = 0000	N/A	lompat bila CX berisi nol	tidak

Perbandingan

CMP : compare. Membandingkan dua buah operand. Hasilnya mempengaruhi sejumlah flag register.
sintaks: CMP {operand 1}, {operand 2}. Operand ini bisa register dengan register , register dengan isi memori, atau register dengan angka.
CMP tidak bisa membandingkan isi memori dengan isi memori. Hasilnya adalah:

Kasus	Bila operand 1 < operand 2	Bila operand 1 = operand 2	Bila operand 1 > operand 2
Signed binary	OF = 1, SF = 1, ZF = 0	OF = 0, SF = 0, ZF = 1	OF = 0, SF = 0, ZF = 0
Unsigned binary	CF = 1, ZF = 0	CF = 0, ZF = 1	CF = 0, ZF = 0