Nama : Anisa Ghoyatul Firdaus

NIM : L200180135

Kelas : D

Modul 1

1. Pengertian ASCII dan tabelnya :
   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. Kode ASCII sebenarnya memiliki komposisi bilangan biner sebanyak 7 bit. Namun, ASCII disimpan sebagai sandi 8 bit dengan menambakan satu angka 0 sebagai bit significant paling tinggi. Bit tambahan ini sering digunakan untuk uji paritas. Karakter control pada ASCII dibedakan menjadi 5 kelompok sesuai dengan penggunaan yaitu berturut-turut meliputi logical communication, Device control, Information separator, Code extention, dan physical communication

Binary	Oct	Dec	Hex	Glyph
010 0000	040	32	20	sp
010 0001	041	33	21	!
010 0010	042	34	22	"
010 0011	043	35	23	#
010 0100	044	36	24	$
010 0101	045	37	25	%
010 0110	046	38	26	&
010 0111	047	39	27	'
010 1000	050	40	28	(
010 1001	051	41	29	)
010 1010	052	42	2A	*
010 1011	053	43	2B	+
010 1100	054	44	2C	,
010 1101	055	45	2D	-
010 1110	056	46	2E	.
010 1111	057	47	2F	/
011 0000	060	48	30	0
011 0001	061	49	31	1
011 0010	062	50	32	2
011 0011	063	51	33	3
011 0100	064	52	34	4
011 0101	065	53	35	5
011 0110	066	54	36	6
011 0111	067	55	37	7
011 1000	070	56	38	8
011 1001	071	57	39	9
011 1010	072	58	3A	:
011 1011	073	59	3B	;
011 1100	074	60	3C	<
011 1101	075	61	3D	=
011 1110	076	62	3E	>
011 1111	077	63	3F	?

Binary	Oct	Dec	Hex	Glyph
100 0000	100	64	40	@
100 0001	101	65	41	A
100 0010	102	66	42	B
100 0011	103	67	43	C
100 0100	104	68	44	D
100 0101	105	69	45	E
100 0110	106	70	46	F
100 0111	107	71	47	G
100 1000	110	72	48	H
100 1001	111	73	49	I
100 1010	112	74	4A	J
100 1011	113	75	4B	K
100 1100	114	76	4C	L
100 1101	115	77	4D	M
100 1110	116	78	4E	N
100 1111	117	79	4F	O
101 0000	120	80	50	P
101 0001	121	81	51	Q
101 0010	122	82	52	R
101 0011	123	83	53	S
101 0100	124	84	54	T
101 0101	125	85	55	U
101 0110	126	86	56	V
101 0111	127	87	57	W
101 1000	130	88	58	X
101 1001	131	89	59	Y
101 1010	132	90	5A	Z
101 1011	133	91	5B	[
101 1100	134	92	5C	\
101 1101	135	93	5D	]
101 1110	136	94	5E	^
101 1111	137	95	5F	_

Binary	Oct	Dec	Hex	Glyph
110 0000	140	96	60	`
110 0001	141	97	61	a
110 0010	142	98	62	b
110 0011	143	99	63	c
110 0100	144	100	64	d
110 0101	145	101	65	e
110 0110	146	102	66	f
110 0111	147	103	67	g
110 1000	150	104	68	h
110 1001	151	105	69	i
110 1010	152	106	6A	j
110 1011	153	107	6B	k
110 1100	154	108	6C	l
110 1101	155	109	6D	m
110 1110	156	110	6E	n
110 1111	157	111	6F	o
111 0000	160	112	70	p
111 0001	161	113	71	q
111 0010	162	114	72	r
111 0011	163	115	73	s
111 0100	164	116	74	t
111 0101	165	117	75	u
111 0110	166	118	76	v
111 0111	167	119	77	w
111 1000	170	120	78	x
111 1001	171	121	79	y
111 1010	172	122	7A	z
111 1011	173	123	7B	{
111 1100	174	124	7C	|
111 1101	175	125	7D	}
111 1110	176	126	7E	~

2. Bahasa Assembly keluarga Intel x86

Dalam program bahasa assembly terdapat 2 jenis yang kita tulis dalam program :

a. **Assembly Directive** (yaitu merupakan kode yang menjadi arahan bagi assembler/compiler untuk menata program).

Assembly Directive	Keterangan
EQU	Pendefinisian konstanta
DB	Pendefinisian data dengan ukuran satuan 1 byte
DW	Pendefinisian data dengan ukuran satuan 1 word
DBIT	Pendefinisian data dengan ukuran satuan 1 bit
DS	Pemesanan tempat penyimpanan data di RAM
ORG	Inisialisasi alamat mulai program
END	Penanda akhir program
CSEG	Penanda penempatan di code segment
XSEG	Penanda penempatan di external data segment
DSEG	Penanda penempatan di internal direct data segment
ISEG	Penanda penempatan di internal indirect data segment
BSEG	Penanda penempatan di bit data segment
CODE	Penanda mulai pendefinisian program
XDATA	Pendefinisian external data
DATA	Pendefinisian internal direct data
IDATA	Pendefinisian internal indirect data
BIT	Pendefinisian data bit
#INCLUDE	Mengikutsertakan file program lain

b. **Instruksi** (yaitu kode yang harus dieksekusi oleh CPU mikrokontroler dengan melakukan operasi tertentu sesuai dengan daftar yang sudah tertanam dalam CPU).

Instruksi	Keterangan Singkatan
ACALL	Absolute Call
ADD	Add
ADDC	Add with Carry
AJMP	Absolute Jump
ANL	AND Logic
CJNE	Compare and Jump if Not Equal
CLR	Clear
CPL	Complement
DA	Decimal Adjust
DEC	Decrement
DIV	Divide
DJNZ	Decrement and Jump if Not Zero
INC	Increment
JB	Jump if Bit Set
JBC	Jump if Bit Set and Clear Bit
JC	Jump if Carry Set
JMP	Jump to Address
JNB	Jump if Not Bit Set
JNC	Jump if Carry Not Set
JNZ	Jump if Accumulator Not Zero
JZ	Jump if Accumulator Zero
LCALL	Long Call
LJMP	Long Jump
MOV	Move from Memory
MOVC	Move from Code Memory
MOVX	Move from Extended Memory
MUL	Multiply
NOP	No Operation
ORL	OR Logic
POP	Pop Value From Stack
PUSH	Push Value Onto Stack
RET	Return From Subroutine
RETI	Return From Interrupt
RL	Rotate Left
RLC	Rotate Left through Carry
RR	Rotate Right
RRC	Rotate Right through Carry
SETB	Set Bit
SJMP	Short Jump
SUBB	Subtract With Borrow
SWAP	Swap Nibbles
XCH	Exchange Bytes
XCHD	Exchange Digits
XRL	Exclusive OR Logic

Keterangan :

1) ACALL (Absolute Call)
   Instruksi ACALL digunakan untuk memanggil sub rutin program
   Contoh :
   START:
   ACALL TUNDA ; Panggil Procedure penundaan waktu
   ….
   TUNDA: ; Label Tunda
   MOV R7,#0FFH ; Isikan Register 7 dengan data 0FFH(255
2) ADD (Add Immediate Data)
   Instruksi ini akan menambah 8 bit data langsung ke dalam isi akumulator dan menyimpan hasilnya pada akumulator.
   Contoh : Add A, #data
   Add A, #@R1 ; Add indirect address
   Add A, R6 ; Add register
   Add A, 30H ; add memori
3) ADDC (Add Carry Plus Immediate Data to Accumulator)
   Instruksi ini akan menambahkan isi carry flag (0 atau 1) ke dalam isi akumulator. Data langsung 8 bit ditambahkan ke akumulator
   Contoh: ADDC A,#012H
   ADDC A, @R1 ; add carry plues indirect address
4) AJMP (Absolute Jump)
   AJMP adalah perintah jump mutlak. Jump dalam 2 KB dimulai dari alamat yang mengikuti perintah AJMP. Ini 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)
   Instruksi ini mengAND-kan isi alamat data dengan isi akumulator
   Contoh : ANL A,#0001000B ;data acumulator akan dikalikan dengan 00010000 Biner
   ANL A, 57H,#01H; logical AND immediate data ke memori
6) CJNE (Compare Indirect Address to Immediate Data)
   Instruksi ini akan 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
   Contoh:
   CJNE R7,#001H,Command0
   MOV A,StepControl
   AJMP Command1
7) CLR (Clear Accumulator)
   Instruksi CLR akan mereset data akumulator menjadi 00H.
   Format : CLR A

8) CPL (Complement Accumulator)
Instruksi CPL akan mengkomplemen isi akumulator
Contoh : CPL A
CPL C ; mengkomplemen isi carry flag

9) DA (Decimal Adjust Accumulator)
Instruksi DA akan mengatur isi akumulator ke padanan BCD, steleah penambahan dua angka BCD.

10) DEC (Decrement Indirect Address)
Instruksi DEC akan mengurangi isi lokasi memori yang ditujukan oleh register R dengan 1, dan hasilnya disimpan pada lokasi tersebut.
Contoh: DEC 40H
DEC R7 ; decrement register

11) DIV (Divide Accumulator by B)
Instruksi DIV akan membagi isi akumulator dengan isi register B. Akumulator berisi hasil bagi, register B berisi sisa pembagian.
Contoh : MOV B,#1H
DIV AB

12) DJNZ (Decrement Register And Jump Id Not Zero)
Instruksi DJNZ akan mengurangi nilai register dengan 1 dan jika hasilnya sudah 0 maka instruksi selanjutnya akan dieksekusi. Jika belum 0 akan menuju ke alamat kode.
Format : DJNZ Rr,Alamat Kode

13) INC (Increment Indirect Address)
Instruksi INCi akan menambahkan isi memori dengan 1 dan menyimpannya pada alamat tersebut.
Contoh: INC A
INC R7 ; increment register

14) JB (Jump if Bit is Set)
Instruksi ini akan 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.
Format : JB alamat bit,alamat kode

15) JBC (Jump if Bit Set and Clear Bit)
Bit JBC, 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. Perintah tersebut juga membersihkan LSB dari akumulator.

16) JC (Jump if Carry is Set)
Instruksi JC akan 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 untuk memerintahkan loncat kesuato alamat kode tertentu.
Format : JMP alamat kode.
Contoh :

Loop:
…
RL A ; Geser data Akumulator Ke kiri
ACALL Long_Delay ; Panggil Procedure penundaan waktu
JMP Loop ; Loncat Ke Procedure Loop

18) JNB (Jump if Bit is Not Set)
Instruksi JNB akan 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)
Perintah JNC menguji bit Carry, dan jika tidak di-set, maka sebuah lompatan akan dilakukan ke alamat relatif yang telah ditentukan. Sebagai contoh, perintah berikut akan menyebabkan loop tanpa henti, karena Carry dibersihkan oleh perintah CLR C dan JNC akan selalu menyebabkan lompatan ke lokasi yang sama yang berlabel 'LOOP'.

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)
Perintah ini menguji konten-konten akumulator. Jika bukan nol, maka lompatan dilakukan ke alamat relatif yang ditentukan dalam perintah.

22) LCALL (Long Call)
LCALL memungkinkan panggilan ke subrutin yang berlokasi dimanapun dalam memori program 64K. Operasi LCALL berjalan seperti berikut:
(i) Menambahkan ke dalam konter program sebanyak 3, karena perintahnya adalah perintah 3-byte.
(ii) Menambahkan penunjuk stack sebanyak 1.
(iii) Menyimpan byte yang lebih rendah dari konter program ke dalam stack.
(iv) Menambahkan penunjuk stack.
(v) Menyimpan byte yang lebih tinggi dari program ke dalam stack.
(vi) Me-load konter program dengan alamat tujuan 16-bit.
Sebagai contoh, jika penunjuk stack dimulai pada 54H, maka perintah LCALL 0400H di lokasi 0100H dalam memori program akan menghasilkan nilai SP, PC dan stack,.

23) LJMP (Long Jump)
Long Jump 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. Sebagai contoh, untuk melompat ke lokasi 0200H, kita dapat menulis "LJMP 0200H".

24) MOV (Move From Memory)
Instruksi ini untuk memindahkan isi akumulator/register atau data dari nilai luar atau alamat lain.
Contoh :
MOV A,#40H
MOV @RO,A
MOV A, P3
MOV C, P1.0
MOV DPTR, #20H
MOVC A, @A+DPTR ; pindahkan code memory offset dari data pointer ke A
MOVX @DPTR, A ; Pindahkan akumulator ke memoeri eksterlah yang dialamati oleh
; Data pointer

25) MOVC (Move From Codec Memory)
Fungsi : Mengisikan (Move) Byte Kode (Code) atau Byte Konstanta (Constant). Instruksi MOVC akan 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)
Instruksi POP akan memindahkan isi akumulator ke memori data eksternal yang alamatnya ditunjukkan oleh isi data pointer.
Contoh: MOVX @DPTR,A

27) MUL (Multiply)
Fungsi : Perkalian (Multiply). MUL AB akan 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 : Tidak Ada (No) Operasi. 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 sebagai instruksi Gerbang logika OR yang akan menjumlahkan Accumulator terhadap nilai yang ditentukan.
Format : ORL A,#data
Contoh : ORL A,#0001000B
Ini berarti data acumulator akan dijumlahkan dengan 00010000 Biner
Jika A bernilai 11100001 maka hasilnya adalah 11110001

30) POP (Pop Stack to Memory)
Instruksi POP akan menempatkan byte yang ditunjukkan oleh stack pointer ke suatu alamat data.
Contoh: POP PSW
POP 03H

31) PUSH (Push Memory onto Stack)
Instruksi ini akan menaikkan stack pointer kemudian menyimpan isinya ke suatu alamat data pada lokasi yang ditunuk oleh stack pointer.
Contoh: PUSH 30H

32) RET (Return from subroutine)
Intruksi untuk kembali dari suatu subrutin program ke alamat terakhir subrutin tersebut di panggil.

33) RETI (Return From Interrupt)
Fungsi : Kembali (Return) dari Interrupt. RETI akan 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 akan memutar setiap bit dalam akumulator satu posisi ke kiri.
Contoh : RL A
RLC A ; rotasi akumulator dan carry flag ke kiri
RR A ; rotasi akumulator ke kanan

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 akan menset carry flag
Contoh : SETB C

39) SJMP (Short Jump)
Sebuah Short Jump 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.