Nama : SekarAndini Khairunnisa

NIM : L200180188

Kelas : E

Tugas Praktikum Modul 1

1. Pengertian dan tabel

PENGERTIAN DAN FUNGSI KODE ASCII

ASCII merupakan kepanjangan dari *(American Standard Code for Information Interchange)*, dan pengertian dari ASCII sendiri adalah suatu standar internasional dalam kode huruf dan simbol seperti Hex dan Unicode tetapi ASCII lebih bersifat universal, contohnya 124 adalah untuk karakter "|". Ia selalu digunakan oleh komputer dan alat komunikasi lain untuk menunjukkan teks.

sedangkan fungsi dari kode ASCII ialah digunakan untuk mewakili karakter-karakter angka maupun huruf didalam komputer, sebagai contoh dapat kita lihat pada karakter 1, 2, 3, A, B, C, dan sebagainya.

Jumlah kode ASCII adalah 255 kode. Kode ASCII 0..127 merupakan kode ASCII untuk manipulasi teks; sedangkan kode ASCII 128..255 merupakan kode ASCII untuk manipulasi grafik. Kode ASCII sendiri dapat dikelompokkan lagi kedalam beberapa bagian:

- Kode yang tidak terlihat simbolnya seperti Kode 10(Line Feed), 13(Carriage Return), 8(Tab), 32(Space)
- Kode yang terlihat simbolnya seperti abjad (A..Z), numerik (0..9), karakter khusus (~!@#$%^&*()_+?:”{})
- Kode yang tidak ada di keyboard namun dapat ditampilkan. Kode ini umumnya untuk kode-kode grafik.

Dalam pengkodean kode ASCII memanfaatkan 8 bit. Pada saat ini kode ASCII telah tergantikan oleh kode UNICODE (Universal Code). UNICODE dalam pengkodeannya memanfaatkan 16 bit sehingga memungkinkan untuk menyimpan kode-kode lainnya seperti kode bahasa Jepang, Cina, Thailand dan sebagainya.

Pada papan keyboard, aktifkan numlock, tekan tombol ALT secara bersamaan dengan kode karakter maka akan dihasilkan karakter tertentu. Misalnya: ALT + 44 maka akan muncul karakter koma (,). Mengetahui kode-kode ASCII sangat bermanfaat misalnya untuk membuat karakter-karakter tertentu yang tidak ada di keyboard.

berikut adalah karakter ASCII

Tabel berikut berisi karakter-karakter ASCII . Dalam sistem operasi Windows dan MS-DOS, pengguna dapat menggunakan karakter ASCII dengan menekan tombol Alt+[nomor nilai ANSI (desimal)]. Sebagai contoh, tekan kombinasi tombol Alt+87 untuk karakter huruf latin "W" kapital.

Sumber:
http://id.wikipedia.org/wiki

Berikut tabel untuk kode ASCII nya:

Tabel ASCII 8 Bit

Char	ASCII Code	Binary	Char	ASCII Code	Binary
a	097	01100001	A	065	01000001
b	098	01100010	B	066	01000010
c	099	01100011	C	067	01000011
d	100	01100100	D	068	01000100
e	101	01100101	E	069	01000101
f	102	01100110	F	070	01000110
g	103	01100111	G	071	01000111
h	104	01101000	H	072	01001000
i	105	01101001	I	073	01001001
j	106	01101010	J	074	01001010
k	107	01101011	K	075	01001011
l	108	01101100	L	076	01001100
m	109	01101101	M	077	01001101
n	110	01101110	N	078	01001110
o	111	01101111	O	079	01001111
p	112	01110000	P	080	01010000
q	113	01110001	Q	081	01010001
r	114	01110010	R	082	01010010
s	115	01110011	S	083	01010011
t	116	01110100	T	084	01010100
u	117	01110101	U	085	01010101
v	118	01110110	V	086	01010110
w	119	01110111	W	087	01010111
x	120	01111000	X	088	01011000
y	121	01111001	Y	089	01011001
z	122	01111010	Z	090	01011010

2. Carilah Daftar bahasa Assembly

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

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

Daftar Assembly Directive

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

Daftar Instruksi

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

untuk yang lebih jelas dan detil:

Perpindahan data

MOV : move. Memindahkan suatu nilai dari register ke memori, memori ke register, atau register ke register.

sintaks: MOV {tujuan}, {sumber}

contoh:

mov AX, 4C00h ;mengisi register AX dengan 4C00(hex).

mov BX, AX ;menyalin isi AX ke BX. mov CL, [BX] ;mengisi register CL dengan data di memori yang alamatnya ditunjuk BX.

mov CL, [BX] + 2 ;mengisi CL dengan data di memori yang alamatnya ditunjuk BX lalu geser maju 2 byte.

mov [BX], AX ;menyimpan nilai AX pada tempat di memori yang ditunjuk BX. mov [BX] – 1, 00101110b

;menyimpan 00101110(bin) pada alamat yang ditunjuk BX lalu geser mundur 1 byte.

LEA : load effective address. Mengisi suatu register dengan alamat offset sebuah data.

sintaks: LEA {register}, {sumber} contoh: lea DX, teks1 XCHG : exchange. Menukar dua buah register langsung.

sintaks: XCHG {register 1}, {register 2} Kedua register harus punya ukuran yang sama.

Bila sama-sama 8 bit (misalnya AH dengan BL) atau sama-sama 16 bit (misalnya CX dan DX),

maka pertukaran bisa dilakukan. Sebenarnya masih banyak perintah perpindahan data,

misalnya IN, OUT, LODS, LODSB, LODSW, MOVS, MOVSB, MOVSW, LDS, LES, LAHF, SAHF, dan XLAT.

Operasi logika

AND : melakukan bitwise and. sintaks: AND {register}, {angka} AND {register 1}, {register 2} hasil disimpan di register 1.

contoh: mov AL, 00001011b mov AH, 11001000b and AL, AH ;sekarang AL berisi 00001000(bin),

sedangkan AH tidak berubah.

OR : melakukan bitwise or. sintaks: OR {register}, {angka} OR {register 1}, {register 2} hasil disimpan di register 1.

NOT : melakukan bitwise not (*one's complement*) sintaks: NOT {register} hasil disimpan di register itu sendiri.

XOR : melakukan bitwise eksklusif or. sintaks: XOR {register}, {angka} XOR {register 1}, {register 2} hasil disimpan di register 1. Tips: sebuah register yang di-XOR-kan dengan dirinya sendiri akan menjadi berisi nol.

SHL : shift left. Menggeser bit ke kiri. Bit paling kanan diisi nol. sintaks: SHL {register}, {banyaknya}

SHR : shift right. Menggeser bit ke kanan. Bit paling kiri diisi nol. sintaks: SHR {register}, {banyaknya}

ROL : rotate left. Memutar bit ke kiri. Bit paling kiri jadi paling kanan kali ini. sintaks: ROL {register},

{banyaknya} Bila banyaknya rotasi tidak disebutkan, maka nilai yang ada di CL akan digunakan sebagai banyaknya rotasi.

ROR : rotate right. Memutar bit ke kanan. Bit paling kanan jadi paling kiri. sintaks: ROR {register},

{banyaknya} Bila banyaknya rotasi tidak disebutkan, maka nilai yang ada di CL akan digunakan sebagai banyaknya rotasi.

Ada lagi : RCL dan RCR.

Operasi matematika

ADD : add. Menjumlahkan dua buah register.

sintaks: ADD {tujuan}, {sumber} operasi yang terjadi: tujuan = tujuan + sumber.

carry (bila ada) disimpan di CF.

ADC : add with carry. Menjumlahkan dua register dan carry flag (CF).

sintaks: ADC {tujuan}, {sumber} operasi yang terjadi: tujuan = tujuan + sumber + CF.

carry (bila ada lagi) disimpan lagi di CF.

INC : increment. Menjumlah isi sebuah register dengan 1.

Bedanya dengan ADD, perintah INC hanya memakan 1 byte memori sedangkan ADD pakai 3 byte.

sintaks: INC {register}

SUB : substract. Mengurangkan dua buah register.

sintaks: SUB {tujuan}. {sumber} operasi yang terjadi: tujuan = tujuan – sumber.

borrow (bila terjadi) menyebabkan CF bernilai 1.

SBB : substract with borrow. Mengurangkan dua register dan carry flag (CF).

sintaks: SBB {tujuan}, {sumber} operasi yang terjadi: tujuan = tujuan – sumber – CF.

borrow (bila terjadi lagi) menyebabkan CF dan SF (sign flag) bernilai 1.

DEC : decrement. Mengurang isi sebuah register dengan 1.

Jika SUB memakai 3 byte memori, DEC hanya memakai 1 byte. sintaks: DEC {register}

MUL : multiply. Mengalikan register dengan AX atau AH.

sintaks: MUL {sumber} Bila register sumber adalah 8 bit,

maka isi register itu dikali dengan isi AL, kemudian disimpan di AX.

Bila register sumber adalah 16 bit, maka isi register itu dikali dengan isi AX,

kemudian hasilnya disimpan di DX:AX. Maksudnya, DX berisi high order byte-nya, AX berisi low order byte-nya.

IMUL : signed multiply. Sama dengan MUL,

hanya saja IMUL menganggap bit-bit yang ada di register sumber sudah dalam bentuk *two's complement*.

sintaks: IMUL {sumber}

DIV : divide. Membagi AX atau DX:AX dengan sebuah register.

sintaks: DIV {sumber} Bila register sumber adalah 8 bit (misalnya: BL), maka operasi yang terjadi: -AX dibagi BL,

-hasil bagi disimpan di AL, -sisa bagi disimpan di AH.

Bila register sumber adalah 16 bit (misalnya: CX), maka operasi yang terjadi: -DX:AX dibagi CX, -hasil bagi disimpan di AX, -sisa bagi disimpan di DX.

IDIV : signed divide. Sama dengan DIV, hanya saja IDIV menganggap bit-bit yang ada di register sumber sudah dalam bentuk *two's complement*.

sintaks: IDIV {sumber}

NEG : negate. Membuat isi register menjadi negatif (*two's complement*).

Bila mau *one's complement*, gunakan perintah NOT. sintaks: NEG {register} hasil disimpan di register itu sendiri.

Pengulangan

LOOP : loop. Mengulang sebuah proses. Pertama register CX dikurangi satu.

Bila CX sama dengan nol, maka looping berhenti. Bila tidak nol, maka lompat ke label tujuan.

sintaks: LOOP {label tujuan} Tips: isi CX dengan nol untuk mendapat jumlah pengulangan terbanyak.

Karena nol dikurang satu sama dengan -1, atau dalam notasi *two's complement* menjadi FFFF(hex) yang sama dengan 65535(dec).

LOOPE : loop while equal. Melakukan pengulangan selama CX ≠ 0 dan ZF = 1. CX tetap dikurangi 1 sebelum diperiksa.

sintaks: LOOP {label tujuan}

LOOPZ : loop while zero. Identik dengan LOOPE.

LOOPNE : loop while not equal.

Melakukan pengulangan selama CX ≠ 0 dan ZF = 0. CX tetap dikurangi 1 sebelum diperiksa.

sintaks: LOOPNE {label tujuan}

LOOPNZ : loop while not zero. Identik dengan LOOPNE.

REP : repeat. Mengulang perintah sebanyak CX kali. sintaks: REP {perintah assembly} contoh:

mov CX, 05 rep inc BX ;register BX ditambah 1 sebanyak 5x.

REPE : repeat while equal. Mengulang perintah sebanyak CX kali, tetapi pengulangan segera dihentikan bila didapati ZF = 1.

sintaks: REPE {perintah assembly}

REPZ : repeat while zero. Identik dengan REPE.

REPNE : repeat while not equal. Mengulang perintah sebanyak CX kali, tetapi pengulangan segera dihentikan bila didapati ZF = 0.

sintaks: REPNE {perintah assembly}

REPNZ : repeat while not zero. Identik dengan REPNE.

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

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				
JNBE	jump if not below or equal	CF = 0 ∧ ZF = 0	unsigned	lompat bila op 1 > op 2	ya
JB	jump if below				
JNAE	jump if not above or equal	CF = 1 ∧ ZF = 0	unsigned	lompat bila op 1 < op 2	ya
JAE	jump if above or equal	CF = 0 ∨ ZF			
JNB	jump if not below	= 1	unsigned	lompat bila op 1 ≥ op 2	ya
JBE	jump if below or equal	CF = 1 ∨ ZF			
JNA	jump if not above	= 1	unsigned	lompat bila op 1 ≤ op 2	ya
JG	jump if greater				
JNLE	jump if not less or equal	OF = 0 ∧ ZF = 0	signed	lompat bila op 1 > op 2	ya
JGE	jump if greater or equal	OF = 0 ∨ ZF			
JNL	jump if not less than	= 1	signed	lompat bila op 1 ≥ op 2	ya
JL	jump if less than		signed	lompat bila op 1 < op 2	ya

JNGE	jump if not greater or equal	OF = 1 ∧ ZF = 0			
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

Operasi stack

PUSH : push. Menambahkan sesuatu ke stack.

Sesuatu ini harus register berukuran 16 bit (pada 386+ harus 32 bit), tidak boleh angka, tidak boleh alamat memori.

Maka Anda tidak bisa mem-push register 8-bit seperti AH, AL, BH, BL, dan kawan-kawannya.

sintaks: push {register 16-bit sumber}

contoh: push DX push AX Setelah operasi push, register SP (stack pointer) otomatis dikurangi 2 (karena datanya 2 byte).

Makanya, “top” dari stack seakan-akan “tumbuh turun”.

POP : pop. Mengambil sesuatu dari stack.

Sesuatu ini akan disimpan di register tujuan dan harus 16-bit. Maka Anda tidak bisa mem-pop menuju AH, AL, dkk.

sintaks: POP {register 16-bit tujuan}

contoh: POP BX Setelah operasi pop, register SP otomatis ditambah 2 (karena 2 byte), sehingga “top” dari stack “naik” lagi.

Tip: karena register segmen tidak bisa diisi langsung nilainya, Anda bisa menggunakan stack sebagai perantaranya.

Contoh kodenya: mov AX, seg teks1 push AX pop DS

PUSHF : push flags. Mem-push semua isi register flag ke dalam stack.

Biasa dipakai untuk mem*backup* data di register flag sebelum operasi matematika. Sintaks: PUSHF ;(saja).

POPF : pop flags. Lawan dari pushf. Sintaks: POPF ;(saja).

POPA : pop all general-purpose registers.

Adalah ringkasan dari sejumlah perintah dengan urutan:

pop DI pop SI pop BP pop SP pop BX pop DX pop CX pop AX

Urutan sudah ditetapkan seperti itu.

sintaks: POPA ;(saja). Jauh lebih cepat mengetikkan POPA daripada mengetik POP-POP-POP yang banyak itu.

PUSHA : push all general-purpose registers. Lawan dari POPA,

dimana PUSHA adalah singkatan dari sejumlah perintah dengan urutan yang sudah ditetapkan:

push AX push CX push DX push BX push SP push BP push SI push DI

Operasi pada register flag

CLC : clear carry flag. Menjadikan CF = 0. Sintaks: CLC ;(saja).

STC : set carry flag. Menjadikan CF = 1. Sintaks: STC ;(saja).

CMC : complement carry flag. Melakukan operasi NOT pada CF. Yang tadinya 0 menjadi 1, dan sebaliknya.

CLD : clear direction flag. Menjadikan DF = 0. Sintaks: CLD ;(saja).

STD : set direction flag. Menjadikan DF = 1.

CLI : clear interrupt flag. Menjadikan IF = 0, sehingga interrupt ke CPU akan di-disable.

Biasanya perintah CLI diberikan sebelum menjalankan sebuah proses penting yang riskan gagal bila diganggu.

STI : set interrupt flag. Menjadikan IF = 1.

Perintah lainnya

ORG : origin. Mengatur awal dari program (bagian static data).

Analoginya seperti mengatur dimana letak titik (0, 0) pada koordinat Cartesius.

sintaks: ORG {alamat awal}

Pada program COM (program yang berekstensi .com), harus ditulis “ORG 100h” untuk mengatur alamat mulai dari progam pada 0100(hex),

karena dari alamat 0000(hex) sampai 00FF(hex) sudah dipesan oleh sistem operasi (DOS).

INT : interrupt. Menginterupsi prosesor.

Prosesor akan:

1. Membackup data registernya saat itu,

2. Menghentikan apa yang sedang dikerjakannya,

3. Melompat ke bagian interrupt-handler (entah dimana kita tidak tahu, sudah ditentukan BIOS dan DOS),

4. Melakukan interupsi,

5. Mengembalikan data registernya,

6. Meneruskan pekerjaan yang tadi ditunda.

sintaks: INT {nomor interupsi}

IRET : interrupt-handler return.

Kita bisa membuat interrupt-handler sendiri dengan berbagai cara.

Perintah IRET adalah perintah yang menandakan bahwa interrupt-handler kita selesai,

dan prosesor boleh melanjutkan pekerjaan yang tadi tertunda.

CALL : call procedure. Memanggil sebuah prosedur.

sintaks: CALL {label nama prosedur}

RET : return. Tanda selesai prosedur.

Setiap prosedur harus memiliki RET di ujungnya.

sintaks: RET ;(saja)

HLT : halt. Membuat prosesor menjadi tidak aktif.

Prosesor harus mendapat interupsi dari luar atau di-reset supaya aktif kembali.

Jadi, jangan gunakan perintah HLT untuk mengakhiri program!!

Sintaks: HLT ;(saja). NOP : no operation.

Perintah ini memakan 1 byte di memori tetapi tidak menyuruh prosesor melakukan apa-apa selama 3 clock prosesor.

Berikut contoh potongan program untuk melakukan *delay* selama 0,1 detik pada prosesor Intel 80386 yang berkecepatan 16 MHz.

mov ECX, 533333334d ;ini adalah bilangan desimal idle: nop loop idle