LAPORAN PRAKTIKUM

TUGAS 1

PRAKTIKUM SISTEM OPERASI



Nama : Muhammad Ridwan Ardhani

Nim : L200190262

Kelas : G

1. Apa yang dimaksud dengan kode ASCII

Yaitu Kode Standar Amerika untuk Pertukaran Informasi atau sebuah standar internasional dalam pengkodean huruf dan simbol seperti Unicode dan Hex tetapi ASCII lebih bersifat universal.

Tabel:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **DEC** | **OCT** | **HEX** | **BIN** | **Symbol** | **Description** |
| 0 | 000 | 00 | 00000000 | NUL | Null char |
| 1 | 001 | 01 | 00000001 | SOH | Start of Heading |
| 2 | 002 | 02 | 00000010 | STX | Start of Text |
| 3 | 003 | 03 | 00000011 | ETX | End of Text |
| 4 | 004 | 04 | 00000100 | EOT | End of Transmission |
| 5 | 005 | 05 | 00000101 | ENQ | Enquiry |
| 6 | 006 | 06 | 00000110 | ACK | Acknowledgment |
| 7 | 007 | 07 | 00000111 | BEL | Bell |
| 8 | 010 | 08 | 00001000 | BS | Back Space |
| 9 | 011 | 09 | 00001001 | HT | Horizontal Tab |
| 10 | 012 | 0A | 00001010 | LF | Line Feed |
| 11 | 013 | 0B | 00001011 | VT | Vertical Tab |
| 12 | 014 | 0C | 00001100 | FF | Form Feed |
| 13 | 015 | 0D | 00001101 | CR | Carriage Return |
| 14 | 016 | 0E | 00001110 | SO | Shift Out / X-On |
| 15 | 017 | 0F | 00001111 | SI | Shift In / X-Off |
| 16 | 020 | 10 | 00010000 | DLE | Data Line Escape |
| 17 | 021 | 11 | 00010001 | DC1 | Device Control 1 (oft. XON) |
| 18 | 022 | 12 | 00010010 | DC2 | Device Control 2 |
| 19 | 023 | 13 | 00010011 | DC3 | Device Control 3 (oft. XOFF) |
| 20 | 024 | 14 | 00010100 | DC4 | Device Control 4 |
| 21 | 025 | 15 | 00010101 | NAK | Negative Acknowledgement |
| 22 | 026 | 16 | 00010110 | SYN | Synchronous Idle |
| 23 | 027 | 17 | 00010111 | ETB | End of Transmit Block |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 24 | 030 | 18 | 00011000 | CAN | Cancel |
| 25 | 031 | 19 | 00011001 | EM | End of Medium |
| 26 | 032 | 1A | 00011010 | SUB | Substitute |
| 27 | 033 | 1B | 00011011 | ESC | Escape |
| 28 | 034 | 1C | 00011100 | FS | File Separator |
| 29 | 035 | 1D | 00011101 | GS | Group Separator |
| 30 | 036 | 1E | 00011110 | RS | Record Separator |
| 31 | 037 | 1F | 00011111 | US | Unit Separator |
| 32 | 040 | 20 | 00100000 |  | Space |
| 33 | 041 | 21 | 00100001 | ! | Exclamation mark |
| 34 | 042 | 22 | 00100010 | " | Double quotes (or speech marks) |
| 35 | 043 | 23 | 00100011 | # | Number |
| 36 | 044 | 24 | 00100100 | $ | Dollar |
| 37 | 045 | 25 | 00100101 | % | Per cent sign |
| 38 | 046 | 26 | 00100110 | & | Ampersand |
| 39 | 047 | 27 | 00100111 | ' | Single quote |
| 40 | 050 | 28 | 00101000 | ( | Open parenthesis (or open bracket) |
| 41 | 051 | 29 | 00101001 | ) | Close parenthesis (or close bracket) |
| 42 | 052 | 2A | 00101010 | \* | Asterisk |
| 43 | 053 | 2B | 00101011 | + | Plus |
| 44 | 054 | 2C | 00101100 | , | Comma |
| 45 | 055 | 2D | 00101101 | - | Hyphen |
| 46 | 056 | 2E | 00101110 | . | Period, dot or full stop |
| 47 | 057 | 2F | 00101111 | / | Slash or divide |
| 48 | 060 | 30 | 00110000 | 0 | Zero |
| 49 | 061 | 31 | 00110001 | 1 | One |
| 50 | 062 | 32 | 00110010 | 2 | Two |
| 51 | 063 | 33 | 00110011 | 3 | Three |
| 52 | 064 | 34 | 00110100 | 4 | Four |
| 53 | 065 | 35 | 00110101 | 5 | Five |
| 54 | 066 | 36 | 00110110 | 6 | Six |
| 55 | 067 | 37 | 00110111 | 7 | Seven |
| 56 | 070 | 38 | 00111000 | 8 | Eight |
| 57 | 071 | 39 | 00111001 | 9 | Nine |
| 58 | 072 | 3A | 00111010 | : | Colon |
| 59 | 073 | 3B | 00111011 | ; | Semicolon |
| 60 | 074 | 3C | 00111100 | < | Less than (or open angled bracket) |
| 61 | 075 | 3D | 00111101 | = | Equals |
| 62 | 076 | 3E | 00111110 | > | Greater than (or close angled bracket) |
| 63 | 077 | 3F | 00111111 | ? | Question mark |
| 64 | 100 | 40 | 01000000 | @ | At symbol |
| 65 | 101 | 41 | 01000001 | A | Uppercase A |
| 66 | 102 | 42 | 01000010 | B | Uppercase B |
| 67 | 103 | 43 | 01000011 | C | Uppercase C |
| 68 | 104 | 44 | 01000100 | D | Uppercase D |
| 69 | 105 | 45 | 01000101 | E | Uppercase E |
| 70 | 106 | 46 | 01000110 | F | Uppercase F |
| 71 | 107 | 47 | 01000111 | G | Uppercase G |
| 72 | 110 | 48 | 01001000 | H | Uppercase H |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 73 | 111 | 49 | 01001001 | I | Uppercase I |
| 74 | 112 | 4A | 01001010 | J | Uppercase J |
| 75 | 113 | 4B | 01001011 | K | Uppercase K |
| 76 | 114 | 4C | 01001100 | L | Uppercase L |
| 77 | 115 | 4D | 01001101 | M | Uppercase M |
| 78 | 116 | 4E | 01001110 | N | Uppercase N |
| 79 | 117 | 4F | 01001111 | O | Uppercase O |
| 80 | 120 | 50 | 01010000 | P | Uppercase P |
| 81 | 121 | 51 | 01010001 | Q | Uppercase Q |
| 82 | 122 | 52 | 01010010 | R | Uppercase R |
| 83 | 123 | 53 | 01010011 | S | Uppercase S |
| 84 | 124 | 54 | 01010100 | T | Uppercase T |
| 85 | 125 | 55 | 01010101 | U | Uppercase U |
| 86 | 126 | 56 | 01010110 | V | Uppercase V |
| 87 | 127 | 57 | 01010111 | W | Uppercase W |
| 88 | 130 | 58 | 01011000 | X | Uppercase X |
| 89 | 131 | 59 | 01011001 | Y | Uppercase Y |
| 90 | 132 | 5A | 01011010 | Z | Uppercase Z |
| 91 | 133 | 5B | 01011011 | [ | Opening bracket |
| 92 | 134 | 5C | 01011100 | \ | Backslash |
| 93 | 135 | 5D | 01011101 | ] | Closing bracket |
| 94 | 136 | 5E | 01011110 | ^ | Caret - circumflex |
| 95 | 137 | 5F | 01011111 | \_ | Underscore |
| 96 | 140 | 60 | 01100000 | ` | Grave accent |
| 97 | 141 | 61 | 01100001 | a | Lowercase a |
| 98 | 142 | 62 | 01100010 | b | Lowercase b |
| 99 | 143 | 63 | 01100011 | c | Lowercase c |
| 100 | 144 | 64 | 01100100 | d | Lowercase d |
| 101 | 145 | 65 | 01100101 | e | Lowercase e |
| 102 | 146 | 66 | 01100110 | f | Lowercase f |
| 103 | 147 | 67 | 01100111 | g | Lowercase g |
| 104 | 150 | 68 | 01101000 | h | Lowercase h |
| 105 | 151 | 69 | 01101001 | i | Lowercase i |
| 106 | 152 | 6A | 01101010 | j | Lowercase j |
| 107 | 153 | 6B | 01101011 | k | Lowercase k |
| 108 | 154 | 6C | 01101100 | l | Lowercase l |
| 109 | 155 | 6D | 01101101 | m | Lowercase m |
| 110 | 156 | 6E | 01101110 | n | Lowercase n |
| 111 | 157 | 6F | 01101111 | o | Lowercase o |
| 112 | 160 | 70 | 01110000 | p | Lowercase p |
| 113 | 161 | 71 | 01110001 | q | Lowercase q |
| 114 | 162 | 72 | 01110010 | r | Lowercase r |
| 115 | 163 | 73 | 01110011 | s | Lowercase s |
| 116 | 164 | 74 | 01110100 | t | Lowercase t |
| 117 | 165 | 75 | 01110101 | u | Lowercase u |
| 118 | 166 | 76 | 01110110 | v | Lowercase v |
| 119 | 167 | 77 | 01110111 | w | Lowercase w |
| 120 | 170 | 78 | 01111000 | x | Lowercase x |
| 121 | 171 | 79 | 01111001 | y | Lowercase y |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 122 | 172 | 7A | 01111010 | z | Lowercase z |
| 123 | 173 | 7B | 01111011 | { | Opening brace |
| 124 | 174 | 7C | 01111100 | | | Vertical bar |
| 125 | 175 | 7D | 01111101 | } | Closing brace |
| 126 | 176 | 7E | 01111110 | ~ | Equivalency sign - tilde |
| 127 | 177 | 7F | 01111111 |  | Delete |
| 128 | 200 | 80 | 10000000 | € | Euro sign |
| 129 | 201 | 81 | 10000001 |  |  |
| 130 | 202 | 82 | 10000010 | ‚ | Single low-9 quotation mark |
| 131 | 203 | 83 | 10000011 | ƒ | Latin small letter f with hook |
| 132 | 204 | 84 | 10000100 | „ | Double low-9 quotation mark |
| 133 | 205 | 85 | 10000101 | … | Horizontal ellipsis |
| 134 | 206 | 86 | 10000110 | † | Dagger |
| 135 | 207 | 87 | 10000111 | ‡ | Double dagger |
| 136 | 210 | 88 | 10001000 | ˆ | Modifier letter circumflex accent |
| 137 | 211 | 89 | 10001001 | ‰ | Per mille sign |
| 138 | 212 | 8A | 10001010 | Š | Latin capital letter S with caron |
| 139 | 213 | 8B | 10001011 | ‹ | Single left-pointing angle quotation |
| 140 | 214 | 8C | 10001100 | Œ | Latin capital ligature OE |
| 141 | 215 | 8D | 10001101 |  |  |
| 142 | 216 | 8E | 10001110 | Ţ | Latin capital letter Z with caron |
| 143 | 217 | 8F | 10001111 |  |  |
| 144 | 220 | 90 | 10010000 |  |  |
| 145 | 221 | 91 | 10010001 | Ř | Left single quotation mark |
| 146 | 222 | 92 | 10010010 | ř | Right single quotation mark |
| 147 | 223 | 93 | 10010011 | Ŗ | Left double quotation mark |
| 148 | 224 | 94 | 10010100 | ŗ | Right double quotation mark |
| 149 | 225 | 95 | 10010101 | • | Bullet |
| 150 | 226 | 96 | 10010110 | Ŕ | En dash |
| 151 | 227 | 97 | 10010111 | ŕ | Em dash |
| 152 | 230 | 98 | 10011000 | ˜ | Small tilde |
| 153 | 231 | 99 | 10011001 | ™ | Trade mark sign |
| 154 | 232 | 9A | 10011010 | š | Latin small letter S with caron |
| 155 | 233 | 9B | 10011011 | › | Single right-pointing angle quotation  mark |
| 156 | 234 | 9C | 10011100 | œ | Latin small ligature oe |
| 157 | 235 | 9D | 10011101 |  |  |
| 158 | 236 | 9E | 10011110 | ţ | Latin small letter z with caron |
| 159 | 237 | 9F | 10011111 | Ÿ | Latin capital letter Y with diaeresis |
| 160 | 240 | A0 | 10100000 |  | Non-breaking space |
| 161 | 241 | A1 | 10100001 | ¡ | Inverted exclamation mark |
| 162 | 242 | A2 | 10100010 | ¢ | Cent sign |
| 163 | 243 | A3 | 10100011 | £ | Pound sign |
| 164 | 244 | A4 | 10100100 | ¤ | Currency sign |
| 165 | 245 | A5 | 10100101 | ¥ | Yen sign |
| 166 | 246 | A6 | 10100110 | ¦ | Pipe, Broken vertical bar |
| 167 | 247 | A7 | 10100111 | § | Section sign |
| 168 | 250 | A8 | 10101000 | ¨ | Spacing diaeresis - umlaut |
| 169 | 251 | A9 | 10101001 | © | Copyright sign |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 170 | 252 | AA | 10101010 | ª | Feminine ordinal indicator |
| 171 | 253 | AB | 10101011 | « | Left double angle quotes |
| 172 | 254 | AC | 10101100 | ¬ | Not sign |
| 173 | 255 | AD | 10101101 |  | Soft hyphen |
| 174 | 256 | AE | 10101110 | ® | Registered trade mark sign |
| 175 | 257 | AF | 10101111 | ¯ | Spacing macron - overline |
| 176 | 260 | B0 | 10110000 | ° | Degree sign |
| 177 | 261 | B1 | 10110001 | ± | Plus-or-minus sign |
| 178 | 262 | B2 | 10110010 | ² | Superscript two - squared |
| 179 | 263 | B3 | 10110011 | ³ | Superscript three - cubed |
| 180 | 264 | B4 | 10110100 | ´ | Acute accent - spacing acute |
| 181 | 265 | B5 | 10110101 | µ | Micro sign |
| 182 | 266 | B6 | 10110110 | ¶ | Pilcrow sign - paragraph sign |
| 183 | 267 | B7 | 10110111 | · | Middle dot - Georgian comma |
| 184 | 270 | B8 | 10111000 | ¸ | Spacing cedilla |
| 185 | 271 | B9 | 10111001 | ¹ | Superscript one |
| 186 | 272 | BA | 10111010 | º | Masculine ordinal indicator |
| 187 | 273 | BB | 10111011 | » | Right double angle quotes |
| 188 | 274 | BC | 10111100 | ¼ | Fraction one quarter |
| 189 | 275 | BD | 10111101 | ½ | Fraction one half |
| 190 | 276 | BE | 10111110 | ¾ | Fraction three quarters |
| 191 | 277 | BF | 10111111 | ¿ | Inverted question mark |
| 192 | 300 | C0 | 11000000 | À | Latin capital letter A with grave |
| 193 | 301 | C1 | 11000001 | Á | Latin capital letter A with acute |
| 194 | 302 | C2 | 11000010 | Â | Latin capital letter A with circumflex |
| 195 | 303 | C3 | 11000011 | Ã | Latin capital letter A with tilde |
| 196 | 304 | C4 | 11000100 | Ä | Latin capital letter A with diaeresis |
| 197 | 305 | C5 | 11000101 | Å | Latin capital letter A with ring above |
| 198 | 306 | C6 | 11000110 | Æ | Latin capital letter AE |
| 199 | 307 | C7 | 11000111 | Ç | Latin capital letter C with cedilla |
| 200 | 310 | C8 | 11001000 | È | Latin capital letter E with grave |
| 201 | 311 | C9 | 11001001 | É | Latin capital letter E with acute |
| 202 | 312 | CA | 11001010 | Ê | Latin capital letter E with circumflex |
| 203 | 313 | CB | 11001011 | Ë | Latin capital letter E with diaeresis |
| 204 | 314 | CC | 11001100 | Ì | Latin capital letter I with grave |
| 205 | 315 | CD | 11001101 | Í | Latin capital letter I with acute |
| 206 | 316 | CE | 11001110 | Î | Latin capital letter I with circumflex |
| 207 | 317 | CF | 11001111 | Ï | Latin capital letter I with diaeresis |
| 208 | 320 | D0 | 11010000 | Ð | Latin capital letter ETH |
| 209 | 321 | D1 | 11010001 | Ñ | Latin capital letter N with tilde |
| 210 | 322 | D2 | 11010010 | Ò | Latin capital letter O with grave |
| 211 | 323 | D3 | 11010011 | Ó | Latin capital letter O with acute |
| 212 | 324 | D4 | 11010100 | Ô | Latin capital letter O with circumflex |
| 213 | 325 | D5 | 11010101 | Õ | Latin capital letter O with tilde |
| 214 | 326 | D6 | 11010110 | Ö | Latin capital letter O with diaeresis |
| 215 | 327 | D7 | 11010111 | × | Multiplication sign |
| 216 | 330 | D8 | 11011000 | Ø | Latin capital letter O with slash |
| 217 | 331 | D9 | 11011001 | Ù | Latin capital letter U with grave |
| 218 | 332 | DA | 11011010 | Ú | Latin capital letter U with acute |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 219 | 333 | DB | 11011011 | Û | Latin capital letter U with circumflex |
| 220 | 334 | DC | 11011100 | Ü | Latin capital letter U with diaeresis |
| 221 | 335 | DD | 11011101 | Ý | Latin capital letter Y with acute |
| 222 | 336 | DE | 11011110 | Þ | Latin capital letter THORN |
| 223 | 337 | DF | 11011111 | ß | Latin small letter sharp s - ess-zed |
| 224 | 340 | E0 | 11100000 | à | Latin small letter a with grave |
| 225 | 341 | E1 | 11100001 | á | Latin small letter a with acute |
| 226 | 342 | E2 | 11100010 | â | Latin small letter a with circumflex |
| 227 | 343 | E3 | 11100011 | ã | Latin small letter a with tilde |
| 228 | 344 | E4 | 11100100 | ä | Latin small letter a with diaeresis |
| 229 | 345 | E5 | 11100101 | å | Latin small letter a with ring above |
| 230 | 346 | E6 | 11100110 | æ | Latin small letter ae |
| 231 | 347 | E7 | 11100111 | ç | Latin small letter c with cedilla |
| 232 | 350 | E8 | 11101000 | è | Latin small letter e with grave |
| 233 | 351 | E9 | 11101001 | é | Latin small letter e with acute |
| 234 | 352 | EA | 11101010 | ê | Latin small letter e with circumflex |
| 235 | 353 | EB | 11101011 | ë | Latin small letter e with diaeresis |
| 236 | 354 | EC | 11101100 | ì | Latin small letter i with grave |
| 237 | 355 | ED | 11101101 | í | Latin small letter i with acute |
| 238 | 356 | EE | 11101110 | î | Latin small letter i with circumflex |
| 239 | 357 | EF | 11101111 | ï | Latin small letter i with diaeresis |
| 240 | 360 | F0 | 11110000 | ð | Latin small letter eth |
| 241 | 361 | F1 | 11110001 | ñ | Latin small letter n with tilde |
| 242 | 362 | F2 | 11110010 | ò | Latin small letter o with grave |
| 243 | 363 | F3 | 11110011 | ó | Latin small letter o with acute |
| 244 | 364 | F4 | 11110100 | ô | Latin small letter o with circumflex |
| 245 | 365 | F5 | 11110101 | õ | Latin small letter o with tilde |
| 246 | 366 | F6 | 11110110 | ö | Latin small letter o with diaeresis |
| 247 | 367 | F7 | 11110111 | ÷ | Division sign |
| 248 | 370 | F8 | 11111000 | ø | Latin small letter o with slash |
| 249 | 371 | F9 | 11111001 | ù | Latin small letter u with grave |
| 250 | 372 | FA | 11111010 | ú | Latin small letter u with acute |
| 251 | 373 | FB | 11111011 | û | Latin small letter u with circumflex |
| 252 | 374 | FC | 11111100 | ü | Latin small letter u with diaeresis |
| 253 | 375 | FD | 11111101 | ý | Latin small letter y with acute |
| 254 | 376 | FE | 11111110 | þ | Latin small letter thorn |
| 255 | 377 | FF | 11111111 | ÿ | Latin small letter y with diaeresis |

1. 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

**Terbagi menjadi 3 bagian utama yaitu :**

1. Komentar

Komentar diawali dengan tanda titik koma (;).

; ini adalah komentar

2. Label

Label diakhiri dengan tanda titik dua (:).

Contoh: main: ,loop: ,proses: ,keluar:

3. Assembler directives

Directives adalah perintah yang ditujukan kepada assembler ketika sedang menerjemahkan program kita ke bahasa mesin.

Directive dimulai dengan tanda titik. **.model** : memberitahu assembler berapa memori yang akan dipakai oleh program kita.

Ada model tiny, model small, model compact, model medium, model large, dan model huge**.**

**.data** : memberitahu assembler bahwa bagian di bawah ini adalah data program.

**.code** : memberitahu assembler bahwa bagian di bawah ini adalah instruksi program.

**.stack** : memberitahu assembler bahwa program kita memiliki stack.

Program EXE harus punya stack. Kira-kira yang penting itu dulu.

Semua directive yang dikenal assembler adalah: .186 .286 .286c .286p .287 .386 .386c .386p .387 .486 .486p .8086 .8087

.alpha .break .code .const .continue .cref .data .data? .dosseg .else .elseif .endif .endw .err .err1 .err2 .errb

.errdef .errdif .errdifi .erre .erridn .erridni .errnb .errndef .errnz .exit .fardata .fardata? .if .lall .lfcond .list .listall .listif .listmacro

.listmacroall .model .no87 .nocref .nolist .nolistif .nolistmacro .radix .repeat .sall .seq .sfcond .stack

.startup .tfcond .type .until .untilcxz .while .xall .xcref .xlist.

**Definisi data**

**DB** : define bytes. Membentuk data byte demi byte. Data bisa data numerik maupun teks.

catatan: untuk membentuk data string, pada akhir string harus diakhiri tanda dolar ($).

sintaks: {label} DB {data} contoh: teks1 db "Hello world $" **DW** : define words.

Membentuk data word demi word (1 word = 2 byte).

sintaks: {label} DW {data} contoh: kucing dw ?, ?, ? ;mendefinisikan tiga slot 16-bit yang isinya don’t care

(disimbolkan dengan tanda tanya)

**DD** : define double words. Membentuk data doubleword demi doubleword (4 byte).

sintaks: {label} DD {data} **EQU** : equals. Membentuk konstanta. sintaks: {label} EQU {data}

contoh: sepuluh EQU 10

Ada assembly yang melibatkan bilangan pecahan (floating point), bilangan bulat (integer), DF (define far words),

DQ (define quad words), dan DT (define ten bytes).

**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. 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 |

**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 |

**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*

**Buat 5 contoh program dalam bahasa assembly**

**1. Program assembly pertama**

.model small

.code

org 100h

main :

mov AH, 02

mov DL, 65

int 21h

int 20h

end main

Simpan sebagai ‘HurufA.asm’. Kemudian assemble file tadi dengan assembler, misalnya TASM (turbo assembler).

Program kita mengeluarkan output  huruf A di layar.

**2. Program assembly ke 2**

Program ini akan menampilkan sebuah string pada console jika dijalankan.

Berikut ini adalah source codenya:

.386  
.MODEL TINY

CSEG SEGMENT PARA PUBLIC USE16 'CODE'  
ASSUME CS:CSEG

org 100h  
start:

;;Routine utama  
  
pushfd ;save flag  
pushad ;save isi semua GPR

;; Menampilkan karakter dilakukan melalui interrupt 10h, service 0Eh  
;; dari bios video card, karakter yang akan ditampilkan diberikan  
;; melalui register si

lea si,msg

mov ah,0Eh ;gunakan service 0Eh  
mov bl,07h ;warna foreground  
xor bh,bh ;gunakan page 0, akan muncul bug tanpa ini

MORE\_DIS:mov al,cs:0+[si] ;karakter yang akan ditulis ada pada al  
cmp al,'$' ;'$' menandakan akhir string  
je NO\_MORE\_DIS  
inc si  
int 10h ;panggil int 10h (serice dari video bios)  
jmp MORE\_DIS  
NO\_MORE\_DIS:

popad ;restore semua GPR  
popfd ;restore semua flags  
retn ;return / akhiri program

;;Definisi Variabel  
msg:  
DB 10,13  
DB 10,13  
DB 0,0,0,0,0,0,0,'Your wish is my command',10,13,10,13  
DB 0,0,0,0,0,0,0,'Tweaking your chipset...',10,13,'$'

CSEG ENDS  
end start

Source code di atas adalah source code dalam masm (Microsoft Assembler).

Program di atas akan menghasilkan output:

Your wish is my command

Tweaking your chipset…

**3. Program assembly ke 3**

section .data

string db "Vendor ID adalah 'XXXXXXXXXXXX'",0xA

section .text

global \_start

\_start:

xor eax,eax

cpuid

mov edi,string

mov [edi+25],ebx

mov [edi+29],edx

mov [edi+33],ecx

mov eax,4

mov ebx,1

mov ecx,string

mov edx,41

int 0x80

mov eax,1

xor ebx,ebx

int 0x80

### Program ini menghasilkan output berupa Menampilkan ID Vendor prosesor Dengan Instruksi Assembly X86

**4. Program assembly ke 4**

.MODEL TINY

.CODE

CODE SEGMENT BYTE PUBLIC 'CODE'

ASSUME CS:CODE,DS:CODE

ORG 0100H

MOV AH,9

INT 21H

RET

DB 'HELLO WORLD$'

CODE ENDS

**Program ini akan menampilkan output tulisan "HELLO WORLD"**

**5. Program assembly ke 5**

.486

.MODEL FLAT

.CODE

PUBLIC \_myFunc

\_myFunc PROC

; Subroutine Prologue

push ebp ; Save the old base pointer value.

mov ebp, esp ; Set the new base pointer value.

sub esp, 4 ; Make room for one 4-byte local variable.

push edi ; Save the values of registers that the function

push esi ; will modify. This function uses EDI and ESI.

; (no need to save EBX, EBP, or ESP)

; Subroutine Body

mov eax, [ebp+8] ; Move value of parameter 1 into EAX

mov esi, [ebp+12] ; Move value of parameter 2 into ESI

mov edi, [ebp+16] ; Move value of parameter 3 into EDI

mov [ebp-4], edi ; Move EDI into the local variable

add [ebp-4], esi ; Add ESI into the local variable

add eax, [ebp-4] ; Add the contents of the local variable

; into EAX (final result)

; Subroutine Epilogue

pop esi ; Recover register values

pop edi

mov esp, ebp ; Deallocate local variables

pop ebp ; Restore the caller's base pointer value

ret

\_myFunc ENDP

END

**Arsitektur x86**

 Ada ALU (aritmethic logic unit), ada register, ada I/O ke system bus, dan ada interkoneksi internal CPU itu sendiri.

Prosesor x86 memiliki 5 kelompok register, yaitu:

**1. General Purpose Registers**Adalah register yang bisa dipakai untuk berbagai keperluan.

Pada prosesor 8086 dan 286, register ini besarnya 16 bit. Register yang ada yaitu:

***AX****: accumulator register. Biasanya dipakai untuk menyimpan hasil hitungan matematika dan untuk menentukan service call.*

***BX****: base register. Biasanya dipakai untuk menunjuk indeks alamat memori.*

***CX****: counter register. Biasanya dipakai untuk pengulangan (loop).*

***DX****: data register. Biasanya dipakai untuk menyimpan data keluar/masuk prosesor, serta dipakai di operasi perkalian dan pembagian.*

Register generik ini masing-masing bisa “dipecah” menjadi dua, satu untuk MSB (high-order byte) dan satunya lagi untuk LSB (low-order byte).

Register AX bisa dipecah menjadi AH (AX high) dan AL (AX low), demikian juga ada BH, BL, CH, CL, DH, DL.

Sebagai contoh, bila AX sedang berisi 0110 0111 1101 0011(bin), maka AH berisi 0110 0111(bin) dan AL berisi 1101 0011(bin).

Untuk prosesor 386 ke atas (yang sudah 32 bit), terdapat register EAX (extended AX), EBX, ECX, dan EDX.

Masing-masing kapasitasnya 32 bit.

**2. Segment Registers**adalah register yang tugasnya mencatat blok memori (baca: segmen) yang sedang digunakan.

Bila isinya diubah sembarangan, program bisa kacau.

Register ini kaitannya dengan memori sementara :

***CS****: code segment. Menunjuk segmen memori tempat kode program yang sekarang sedang jalan.*

***DS****: data segment. Menunjuk segmen memori tempat data-data program disimpan (seperti pre-defined string,*

*buffer string, dan konstanta-konstanta).*

***SS****: stack segment. Menunjuk segmen memori tempat “top” dari stack saat ini.*

*“Segmen”nya, bukan “top”nya. Pembahasan tentang stack masih nanti.*

***ES****: extra segment. Adalah register “bonus” yang belum tentu dipakai, tergantung programnya.*

*Misalnya untuk menunjuk alamat video memory pada program game yang fokus pada grafis.*

Pada prosesor 386 ke atas, ada tambahan extra segment lagi yang bernama **FS** dan **GS**.

Huruf “F” dan “G” hanyalah urutan abjad sesudah “E” (ES), tidak ada arti khususnya.

Semua register segmen, baik di prosesor 16-bit maupun 32-bit, kapasitasnya adalah 16-bit.

**3. Pointer Registers** Adalah register yang berfungsi sebagai *pointer*.

Isi dari register ini adalah alamat memori, makanya dia dikatakan “menunjuk” (*to point*) ke suatu alamat memori tertentu.

Kapasitas register pointer adalah 16 bit. Karena hanya 16 bit, maka tentu saja tidak semua alamat memori bisa ditunjuknya.

Besar memori yang mampu ditangani hanya 216 byte = 64 KB (saja), sangat kecil.

Padahal prosesor 16 bit bisa menangani memori sampai 1 MB.

Maka dari itu, register pointer ini bekerja berpasangan dengan register segmen menghasilkan alamat memori lengkap,

dan mampu menangani sampai 1 MB.

***SP****: stack pointer. Menunjuk ke “top” dari stack-nya langsung.*

*Operasi push dan pop akan mengubah isi dari SP. Penjelasan tentang stack masih nanti.*

***BP****: base pointer. Menunjuk ke sebuah alamat di bagian data program.*

*Bekerjasama dengan DS. Biasa dipakai untuk menunjuk elemen array.*

***IP****: instruction pointer. Menunjuk ke alamat memori tempat instruksi berikutnya, di bagian kode program.*

*Anggap saja IP adalah program counter (PC). Register ini diubah otomatis sejalan dengan jalannya program.*

*Pada prosesor 386 ke atas, terdapat register ESP, EBP, dan EIP yang kapasitasnya 32 bit*

*sehingga mampu menangani memori sampai 232 byte = 4 GB.*

**4. Index Registers** Register ini digunakan oleh operasi string dan block transfer di memori.

***SI****: source index.*

***DI****: destination index.*

Kapasitasnya 16 bit. Pada prosesor 386 ke atas, terdapat ESI dan EDI yang kapasitasnya 32 bit.

**5. Flag Registers** Namanya juga “bendera”, register ini berfungsi menandakan suatu keadaan “ya” (mengibarkan bendera)

atau “tidak” (tidak mengibarkan bendera). Tentu saja tidak ada bendera di dalam CPU Anda,

yang ada hanyalah bit 1 atau 0. Register bendera ini masing-masing besarnya 1 bit saja.

***OF****: overflow register. Bila terjadi overflow pada operasi matematika, maka OF bernilai 1. Bila tidak, isi 0.*

***SF****: sign flag. Jika suatu operasi menghasilkan angka negatif, SF berisi 1.*

***ZF****: zero flag. Jika suatu operasi menghasilkan angka nol, ZF berisi 1.*

***CF****: carry flag. Berisi bit carry pada operasi penjumlahan atau bit borrow pada operasi pengurangan.*

***DF****: direction flag. Menunjukkan arah pembacaan byte (maju atau mundur) pada operasi string.*

***PF****: parity flag. Bila angka yang dihitung genap atau ganjil (tergantung sistemnya mau genap apa ganjil), PF berisi 1.*

***AF****: auxiliary flag. Berisi 1 setelah penjumlahan dua bilangan BCD. Fungsinya seperti carry flag (CF) setelah bit ke-4 (bukan ke-8)*

***TF****: trap flag. Menunjukkan mode debugging on atau off. Ini urusan internal CPU.*

***IF****: interrupt flag. Bila IF bernilai 0, maka interupsi ke prosesor akan diabaikan.*

Register yang hanya terdapat di 80286 ke atas:

*NT : nested task.*

*IOPL : I/O protection level. (2 bit)*

*PE : protection enable.*

*MP : monitor co-processor.*

*EM : emulate co-processor.*

*TS : task switched.*

*ET : extention type.*

*RF : resume flag.*

*VF : virtual 8086 mode.*

Jadi itulah register yang ada di prosesor Pentium/AMD (x86).

Lebih sedikit dibanding MIPS karena di MIPS ada 32 register dimana 16 diantaranya adalah general purpose,

masih ditambah lagi 12 register general-purpose untuk floating point. Semuanya 32 bit.

Di x86, register general purpose cuma 4, 16 bit saja.

Makanya ketika mendesain program, manfaatkan sumberdaya prosesor yang terbatas ini sebaik-baiknya.

Manajemen memori

1. Bagian “reserved” sudah diatur oleh sistem operasi dan tidak bisa kita utak-atik lagi.  
   Untuk program COM (program DOS yang berekstensi .com), bagian reserved ini sudah pasti ukurannya dan letaknya,  
   yaitu dari alamat memori 0000(hex) sampai 00FF(hex).
2. Bagian static data berisi data terdefinisi yang tidak berubah,   
   seperti konstanta, string terdefinisi, dan buffer.
3. Bagian program adalah instruksi-instruksi program yang nanti kita tulis dalam bahasa assembly.   
   Tentu yang dijalankan oleh prosesor bukanlah bahasa assembly-nya, tapi sudah dalam bentuk bahasa mesin.
4. Bagian dynamic data, tempat dimana variabel temporer berada.  
   Perintah “malloc” di bahasa C memesan tempat di bagian ini.   
   Dynamic data bisa membesar dan mengecil, tergantung banyaknya data yang diciptakan dan dihapus.
5. Terakhir adalah stack, memori yang pasti disediakan di setiap program.  
   Dasar dari stack terletak di alamat paling tinggi, dan tumbuh turun ke bawah.   
   Nah, “top” dari stack bisa saja tabrakan dengan dynamic data; jika demikian akan muncul error “stack overflow.”

Memori di komputer kita yang hitungannya sudah gigabyte, dibagi-bagi menjadi beberapa blok (dikenal sebagai segmen).

Satu segmen berukuran 64 KB.

Apa bedanya program COM dan EXE? Program COM maksimal memakai memori hanya **satu**segmen,

yaitu 64 KB (makanya kecil sekali). Zaman DOS dulu (dimana memori kecil dan sangat mahal), banyak sekali program COM.

Sebaliknya, program EXE bisa memakai **lebih dari satu** segmen.

Bisa saja static data-nya banyak sekali, sampai disimpan bersegmen-segmen.

Stacknya juga barangkali banyak sekali, memakai puluhan segmen.

Coba jalankan Microsoft Word (winword.exe), lalu buka task manager.

Lihat memori yang dipakainya (di bagian private use working set).

Waktu pertama kali dijalankan, di komputer saya tertampil 14280 KB.

Bagi dengan 64, maka Microsoft Word ini memakai 224 segmen di RAM.

**Bermain dengan alamat memori**

Dikenal dua jenis alamat memori. Yang pertama adalah alamat absolut, yaitu alamat yang secara fisik diakses oleh prosesor.

Yang kedua adalah alamat relatif, yaitu alamat yang didasarkan pada pembagian segmen dan pergeseran (offset).

Misalkan kita kembali ke zaman dulu, dimana prosesor masih 16-bit dan memori komputer masih 1 MB.

Satu megabyte berarti 1024 KB, sama dengan 1.048.576 byte. Setiap byte ini punya alamat absolut masing-masing, yaitu dari 00000(hex) sampai FFFFF(hex).

Komputer membagi memori 1 MB tadi seperti ini:

Segmen 0000(hex) dimulai dari alamat absolut 00000(hex) sampai 0FFFF(hex) (alamat 0 – 65535).

Segmen 0001(hex) dimulai dari alamat absolut 00010(hex) sampai 1000F(hex) (16 – 65551).

Dan seterusnya sampai segmen FFFF(hex).

Cara mengubah alamat relatif ke alamat absolut:

1. Kalikan angka segmen dengan 16(dec) (= 10(hex))

2. Kemudian jumlahkan dengan offset-nya.

Notasi untuk alamat relatif adalah [segmen]:[offset] dalam heksadesimal. Misal 0000:0001 dan FF87:550A.

Pembagian seperti tadi mengakibatkan adanya tumpang tindih (overlap) antarsegmen. Ilustrasinya:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | +0 | +1 | +2 | +3 | +4 | +5 | +6 | +7 | +8 | +9 | +A | +B | +C | +D | +E | +F |
| 0001 | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x |
| 0002 | x | x | x | x | x | x | X | x | x | x | x | x | x | x | x | x |
| 0003 | x | x | x | x | x | x | x | x | x | x | X | x | x | x | x | x |
| 0004 | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x |
| 0005 | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x |

Dalam tabel ini, baris adalah segmen, kolom adalah offset.

Tanda **x** adalah byte yang saat itu ada di memori.

Lihat byte yang saya tandai dengan merah (tanda x warna merah) itu.

Kita boleh bilang bahwa dia terletak di segmen 0002 digeser 6 byte, bukan? Maka alamat relatifnya adalah **0002:0006**.

Tapi kita juga boleh bilang kalau dia terletak di segmen 0001 digeser 22 byte (= 16(hex)) bukan?

Kalau begitu alamatnya menjadi **0001:0016.** Baik 0002:0006 maupun 0001:0016 menunjuk ke alamat absolut yang sama, yaitu **00026(hex)**.

Demikian juga byte yang saya tandai dengan warna hijau. Dia bisa diakses dengan:

*Alamat absolut = 0003A(hex)*

*Alamat relatif #1 = 0003:000A*

*Alamat relatif #2 = 0002:001A*

*Alamat relatif #3 = 0001:002A*

Pada alamat relatif #1, “0003″ disimpan pada register segmen (entah itu SS, DS, CS, atau ES).

Sedangkan “000A” disimpan pada register pointer (entah itu SP, IP, atau BP).

Ilustrasi di atas sebenarnya kurang tepat, karena saya menggambarkan satu segmen sebesar 16 byte saja.

Kenyataannya, satu segmen adalah 64 KB.

Referensi:

<https://oferiachacha.blogspot.com/2012/11/bahasa-assembly-untuk-x86.html>

<http://end4su.wordpress.com/2009/06/18/assembler/>