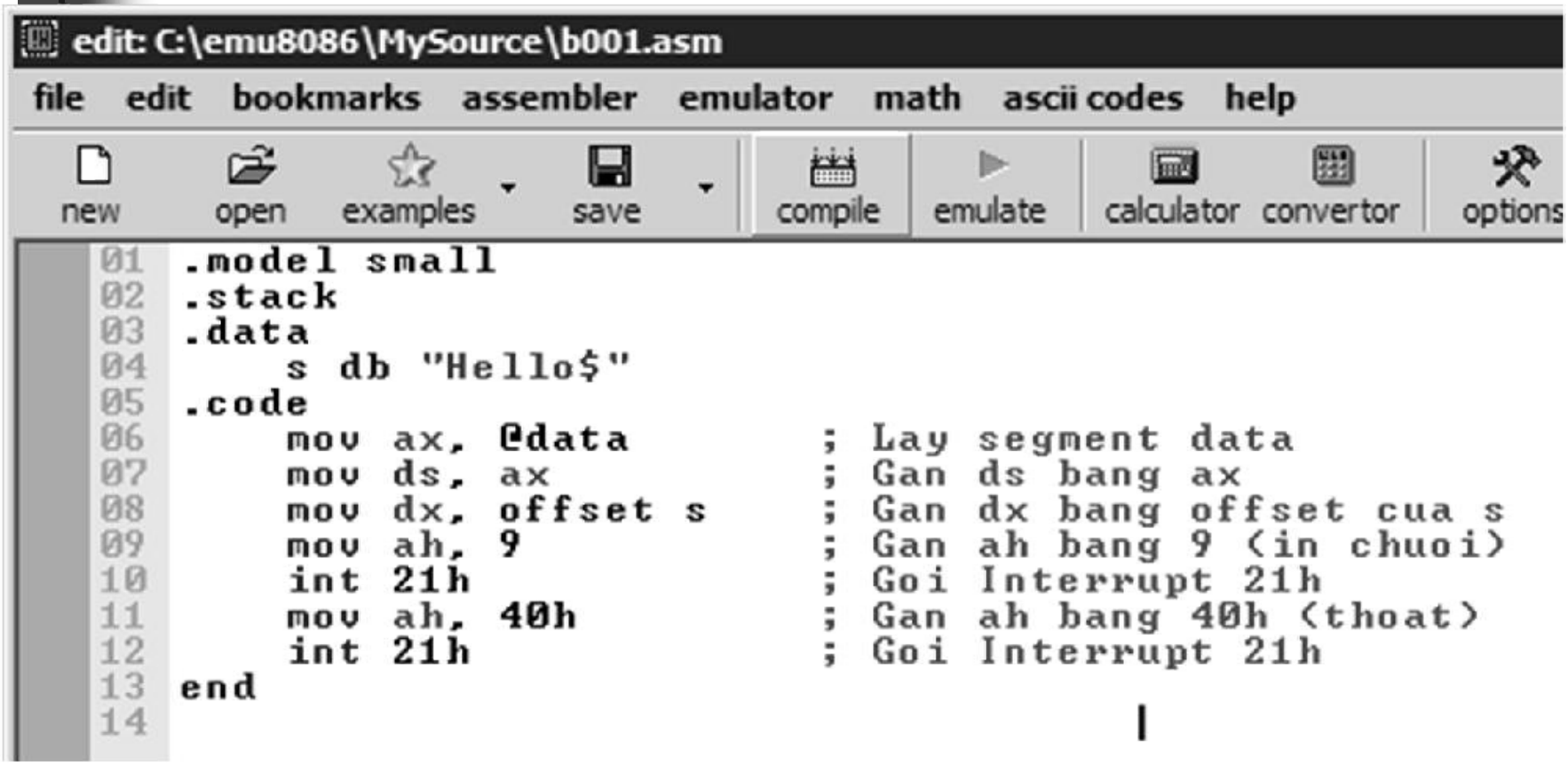
Ch05 - HỢP NGỮ

HỢP NGỮ

```
.model small
.stack
.data
    s db "Hello$"
.code
    mov ax, @data ; Lay segment data
    mov ds, ax    ; Gan ds bang ax
    mov dx, offset s    ; Gan dx bang offset cua s
    mov ah, 9           ; Gan ah bang 9 (in chuoi)
    int 21h       ; Goi Interrupt 21h
    mov ah, 4Ch   ; Gan ah bang 4ch (thoat)
    int 21h       ; Goi Interrupt 21h
end
```

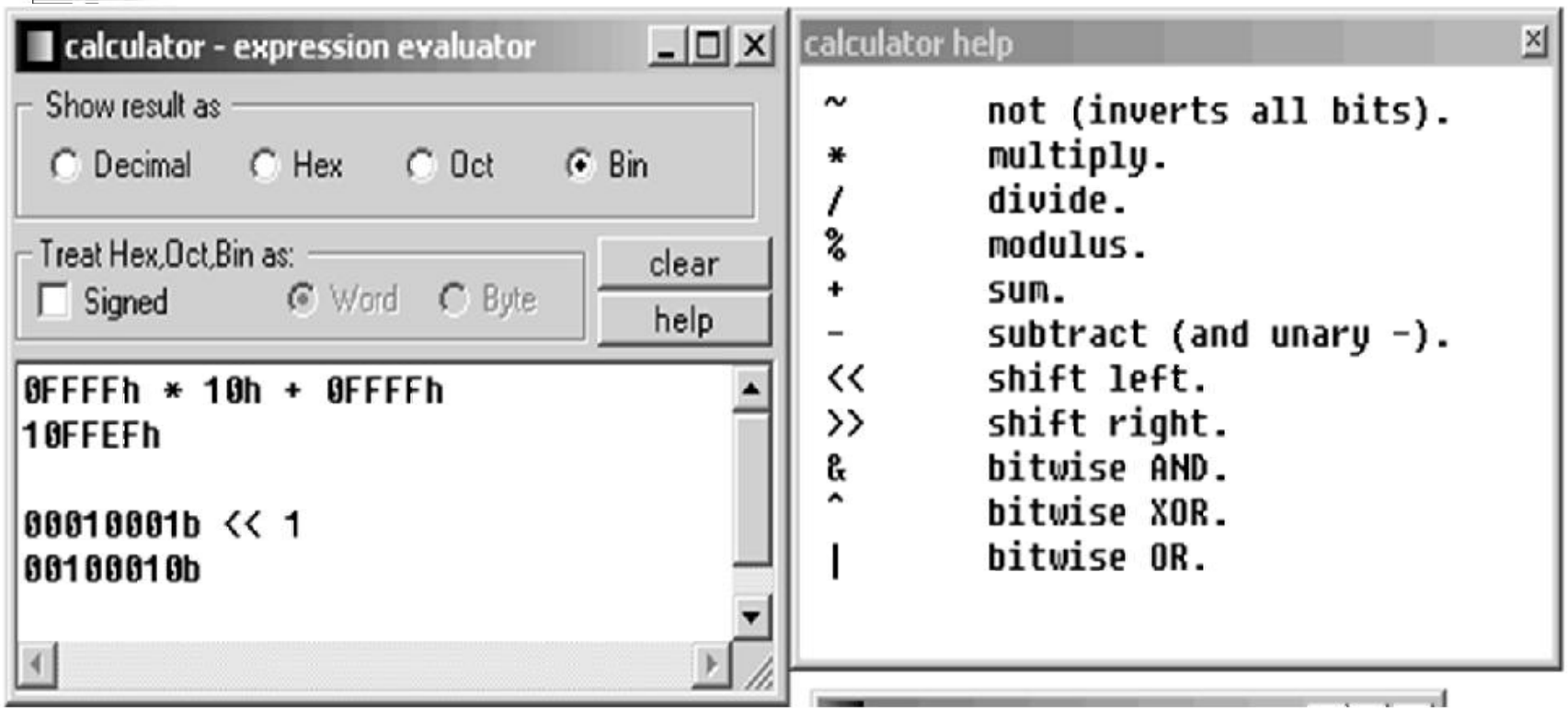
HỢP NGỮ

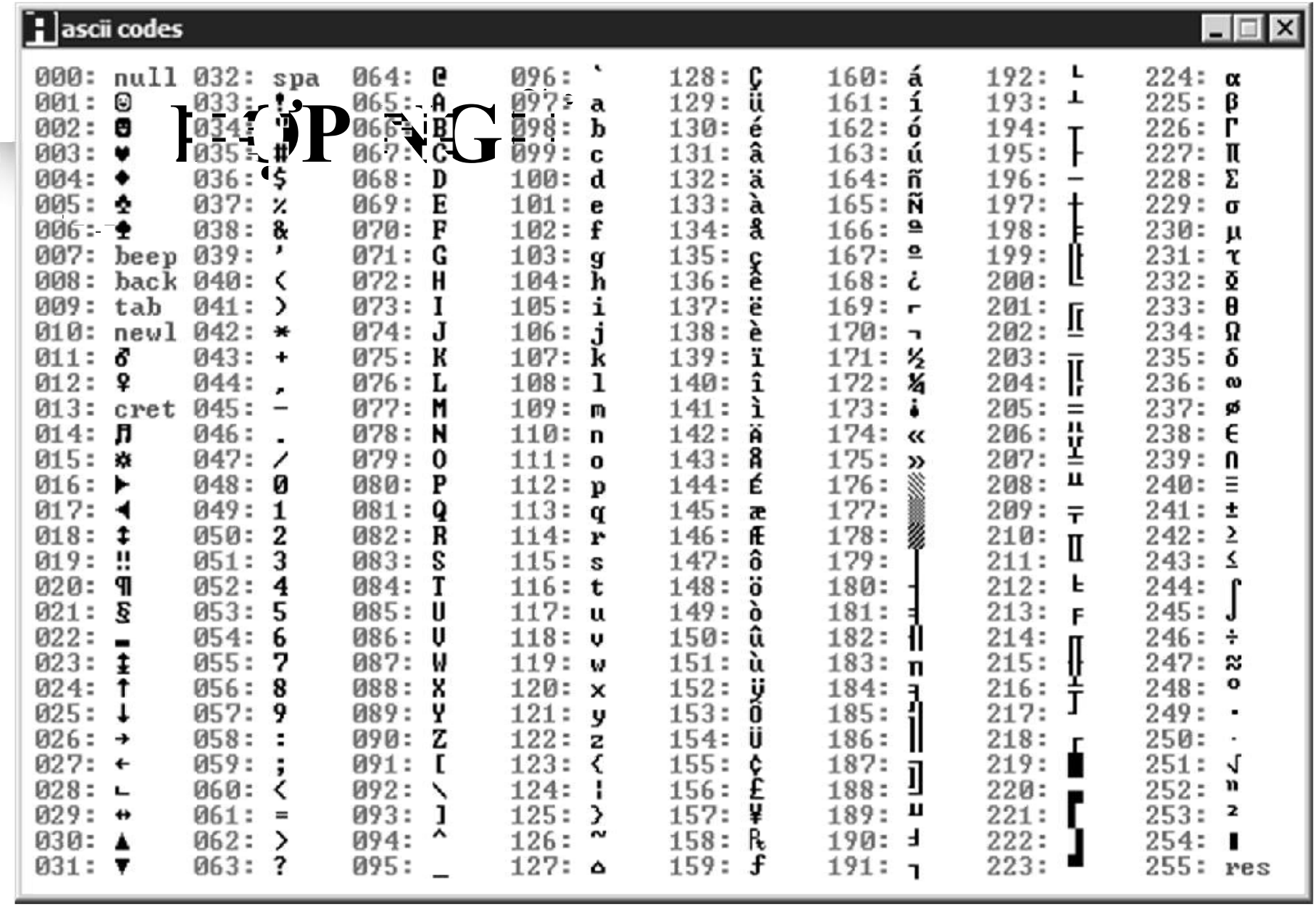

edit: C:\emu8086\MySource\b001.asm
file edit bookmarks assembler emulator math ascii codes help
new open examples save compile emulate calculator convertor options
01 .model small
02 .stack
03 .data
04 s db "Hello$"
05 .code
06 mov ax, @data ; Lay segment data
07 mov ds, ax ; Gan ds bang ax
08 mov dx, offset s ; Gan dx bang offset cua s
09 mov ah, 9 ; Gan ah bang 9 (in chuoi)
10 int 21h ; Goi Interrupt 21h
11 mov ah, 40h ; Gan ah bang 40h (thoat)
12 int 21h ; Goi Interrupt 21h
13 end
14

HỢP NGỮ

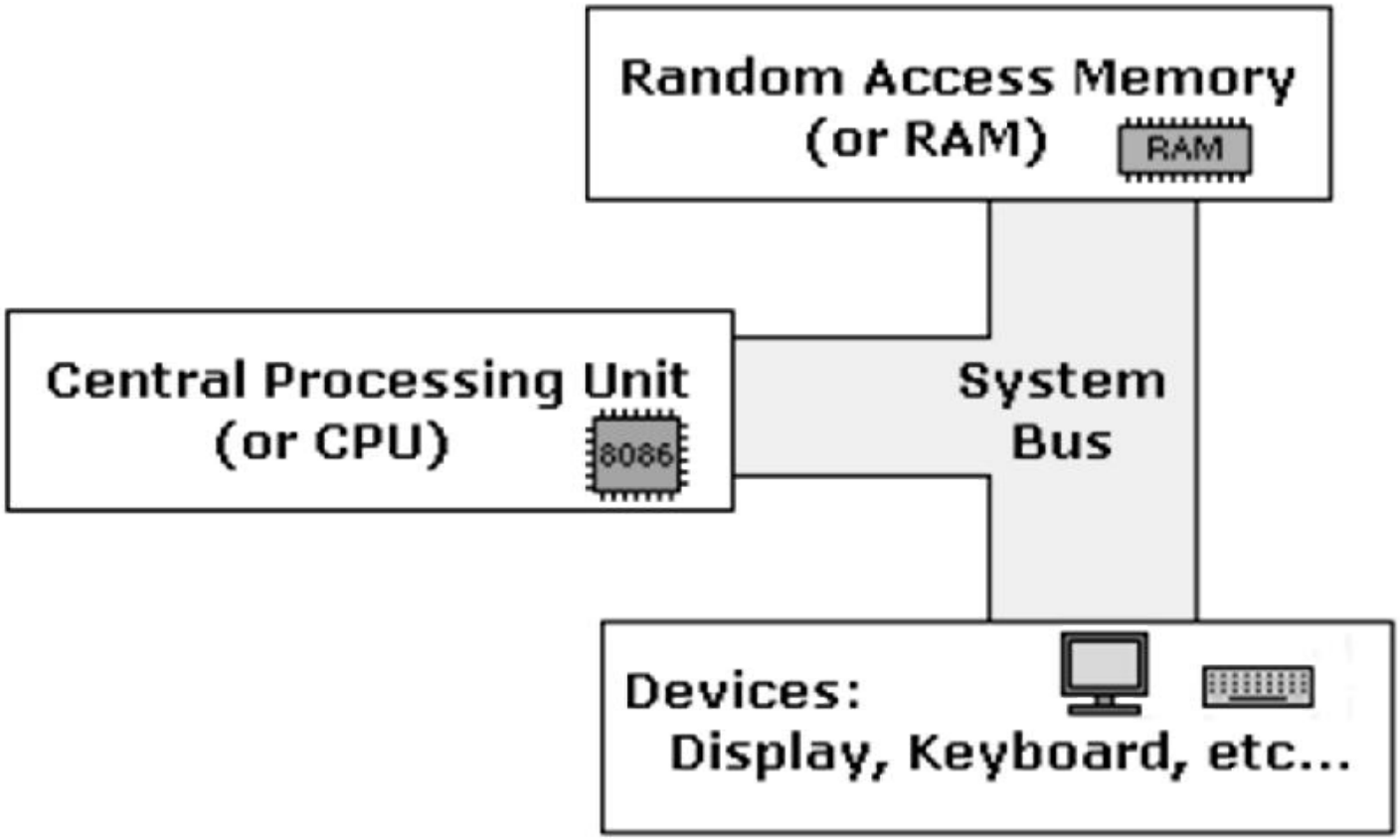

base convertor
8 bit
16 bit
hex:
41
signed
unsigned
dec:
65
65
ascii char:
A
oct:
101
bin:
01000001

HỢP NGỮ

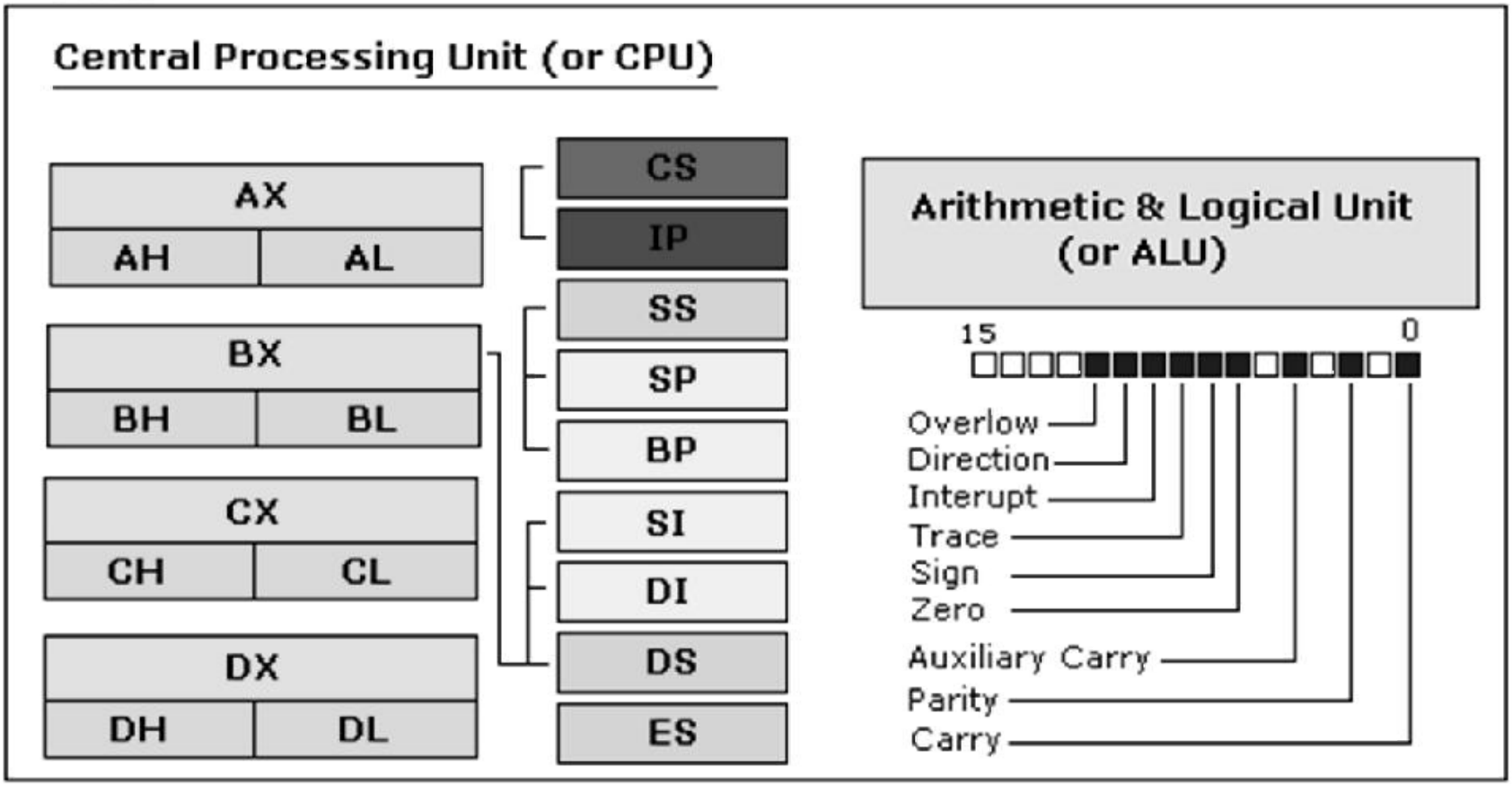

calculator - expression evaluator
Show result as
Decimal
Hex
Oct
Bin
Treat Hex,Oct,Bin as:
Signed
Word
Byte
clear
help
0FFFFh * 10h + 0FFFFh
10FFEFh
00010001b << 1
00100010b
calculator help
~ not (inverts all bits).
* multiply.
/ divide.
% modulus.
+ sum.
- subtract (and unary -).
<< shift left.
>> shift right.
& bitwise AND.
^ bitwise XOR.
| bitwise OR.


ascii codes
HOPING
000: null 032: spa 064: @ 096: ` 128: Ç 160: á 192: └ 224: α
001: ☺ 033: ! 065: A 097: a 129: ü 161: í 193: ┴ 225: ß
002: ☻ 034: " 066: B 098: b 130: é 162: ó 194: ┬ 226: Γ
003: ♥ 035: # 067: C 099: c 131: â 163: ú 195: ├ 227: π
004: ♦ 036: $ 068: D 100: d 132: ä 164: ñ 196: ─ 228: Σ
005: ♣ 037: % 069: E 101: e 133: à 165: Ñ 197: ┼ 229: σ
006: ♠ 038: & 070: F 102: f 134: å 166: ª 198: ╞ 230: µ
007: beep 039: ' 071: G 103: g 135: ç 167: º 199: ╟ 231: τ
008: back 040: (072: H 104: h 136: ê 168: ¿ 200: ╚ 232: Φ
009: tab 041:) 073: I 105: i 137: ë 169: ⌐ 201: ╔ 233: Θ
010: newl 042: * 074: J 106: j 138: è 170: ¬ 202: ╩ 234: Ω
011: ♂ 043: + 075: K 107: k 139: ï 171: ½ 203: ╦ 235: δ
012: ♀ 044: , 076: L 108: l 140: î 172: ¼ 204: ╠ 236: ∞
013: cret 045: - 077: M 109: m 141: ì 173: ¡ 205: ═ 237: ø
014: ♫ 046: . 078: N 110: n 142: Ä 174: « 206: ╬ 238: ε
015: ☼ 047: / 079: O 111: o 143: Å 175: » 207: ╧ 239: ∩
016: ► 048: 0 080: P 112: p 144: É 176: ░ 208: ╨ 240: ≡
017: ◄ 049: 1 081: Q 113: q 145: æ 177: ▒ 209: ╤ 241: ±
018: ↕ 050: 2 082: R 114: r 146: Æ 178: ▓ 210: ╥ 242: ≥
019: ‼ 051: 3 083: S 115: s 147: ô 179: │ 211: ╙ 243: ≤
020: ¶ 052: 4 084: T 116: t 148: ö 180: ┤ 212: ╘ 244: ⌠
021: § 053: 5 085: U 117: u 149: ò 181: ╡ 213: ╒ 245: ⌡
022: ▬ 054: 6 086: V 118: v 150: û 182: ╢ 214: ╓ 246: ÷
023: ↨ 055: 7 087: W 119: w 151: ù 183: ╖ 215: ╫ 247: ≈
024: ↑ 056: 8 088: X 120: x 152: ÿ 184: ╕ 216: ╪ 248: °
025: ↓ 057: 9 089: Y 121: y 153: Ö 185: ╣ 217: ┘ 249: ∙
026: → 058: : 090: Z 122: z 154: Ü 186: ║ 218: ┌ 250: ·
027: ← 059: ; 091: [123: { 155: ¢ 187: ╗ 219: █ 251: √
028: ∟ 060: < 092: \ 124: ¦ 156: £ 188: ╝ 220: ▄ 252: ⁿ
029: ↔ 061: = 093:] 125: } 157: ¥ 189: ╜ 221: ▌ 253: ²
030: ▲ 062: > 094: ^ 126: ~ 158: ₧ 190: ╛ 222: ▐ 254: ■
031: ▼ 063: ? 095: _ 127: ⌂ 159: ƒ 191: ┐ 223: ▀ 255: res

HỢP NGỮ


Random Access Memory
(or RAM)
RAM
Central Processing Unit
(or CPU)
8086
System
Bus
Devices:
Display, Keyboard, etc...

HỢP NGỮ


Central Processing Unit (or CPU)
AX
AH
AL
BX
BH
BL
CX
CH
CL
DX
DH
DL
CS
IP
SS
SP
BP
SI
DI
DS
ES
Arithmetic & Logical Unit
(or ALU)
15
0
Overlow
Direction
Interupt
Trace
Sign
Zero
Auxiliary Carry
Parity
Carry

HỢP NGỮ

- **Memory Access**

[BX + SI] [BX + DI] [BP + SI] [BP + DI]	[SI] [DI] d16 (variable offset only) [BX]	[BX + SI + d8] [BX + DI + d8] [BP + SI + d8] [BP + DI + d8]
[SI + d8] [DI + d8] [BP + d8] [BX + d8]	[BX + SI + d16] [BX + DI + d16] [BP + SI + d16] [BP + DI + d16]	[SI + d16] [DI + d16] [BP + d16] [BX + d16]

HỢP NGỮ

- **Một số lệnh cơ bản:**

MOV dst, src

. dst <- src,

. Dst, src cùng kích thước

mov reg, reg

mov reg, mem

mov reg, imm

mov mem, reg

mov mem, imm

HỢP NGỮ

xchg dst, src

. dst <-> src, tráo đổi giá trị cho nhau

. Dst, src phải cùng kích thước

xchg reg, reg

xchg reg, mem

xchg mem, reg

HỢP NGỮ

mov ax, 513

xchg ah, al

- ⇒ **AX = 513 = 0201h**
- ⇒ **=> AH = 02, AL =01**
- ⇒ **Goi xchg => AL = 02, AH =01**
- ⇒ **=> AH = AH*256+AL = 258**

HỢP NGỮ

ADD dst, src

dst = dst+ src, cộng dồn src vao dst

. Dst, src phải cùng kích thước

add reg, reg

add reg, mem

add reg, imm

add mem, reg

add mem, imm

HỢP NGỮ

mov ax, 456

add al, 56

⇒ **Ah =1, al = 200**

⇒ **Add al, 56 =>200+56 = 0**

⇒ **Ax = ah *256 + al = 1 * 256 +0= 256**

HỢP NGỮ

SUB dst, src

dst = dst - src, trừ bớt dst cho src

. Dst, src phải cùng kích thước

sub reg, reg

sub reg, mem

sub reg, imm

sub mem, reg

sub mem, imm

HỢP NGỮ

mov ax, 256
sub al, 56

⇒ **Ah = 01, al =0**
⇒ **Sub al, 56=> al = -56 =200**
⇒ **Ax = ah *256+al = 1*256+200 = 456**

HỢP NGỮ

MUL src (nhân không dấu)

<u>Nếu src là 8 bit (1byte):</u>

AX = AL *src

=>Chuẩn bị thừa số thứ nhất trong AL

Thừa số thứ hai là src

Kết quả: tích trong AX (1 word = 2byte)

```
mul  reg8
mul  mem8
```

HỢP NGỮ

Nếu src là 16 bit (2 byte):

DX:AX = AX *src

=>Chuẩn bị thừa số thứ nhất trong AX

Thừa số thứ hai là src

Kết quả: tích trong DX:AX (2 word, word cao trong DX, word thấp trong AX)

=>Thanh ghi DX bị thay đổi nội dung.

```
mul  reg16
mul  mem16
```

HỢP NGỮ

DIV src (chia không dấu, lấy phần dư)

Nếu src là 8 bit (1byte):

AL = AX /src (AL : thương)

AH = AX % src (AH: phần dư)

=>Chuẩn bị số bị chia trong AX, số chia là src. Kết quả: AL là thương, AH phần dư

div reg8

div mem8

HỢP NGỮ

DIV src (chia không dấu, lấy phần dư)

Nếu src là 16 bit (2 byte):

AX = (DX:AX) /src (AX : thương)

DX = (DX:AX) % src (DX: phần dư)

=>Chuẩn bị số bị chia: word cao trong DX, WORD thấp trong AX, số chia là src. Kết quả: AX là thương, DX phần dư

div reg16

div mem16

HỢP NGỮ

INC dst

dst = dst +1

inc reg

inc mem

HỢP NGỮ

dec dst

dst = dst -1

dec reg

dec mem

HỢP NGỮ

neg dst (phép đảo dấu, bù 2)

dst = - dst = 2^n – dst (n số bit)

neg reg

neg mem

Mov al, 56

Neg al

=> al = 200

HỢP NGỮ

not dst (phép đảo bit, bù 1)

dst = đảo bít của dst = 2^n – dst-1

not reg

not mem

Mov al, 15 => al = 00001111b

not al => al = 11110000b = 240

=> al = 240

HỢP NGỮ

Nhập ký tự:

```
mov ah, 1
int 21h
```

=> Ký tự nhập trong AL

HỢP NGỮ

Xuất ký tự:

```
mov dl , `A' ; DL chứa ký tự cần xuất
mov ah, 2
int 21h
```

=> xuất ký tự trong DL ra màn hình

HỢP NGỮ

Xuất chuỗi:

```
mov dx, offset  chuoicanxuat
mov ah, 9
int 21h
```

=> xuất chuỗi ra màn hình, DX là offset của chuỗi, chuỗi kết thúc là ký tự $

HỢP NGỮ

Xuất giá trị biến kiểu word

Include 'emu8086.inc'

Mov ax, biến cần xuất giá trị

Call print_num_uns

Trước chỉ thị end kết thúc chương trình:

Define_print_num_uns

HỢP NGỮ

- **Khai báo biến đơn (.data)**

  Khai báo biến kiểu byte (8bit)

  tên_biến **DB** *giá trị khởi tạo*

  Khai báo biến kiểu word (2byte = 16bit)

  tên_biến **DW** *giá trị khởi tạo*

  DB - Define Byte.
  DW - Define Word.

HỢP NGỮ

- **byte ptr**
- **word ptr**

HỢP NGỮ

mov ax, 456

add al, 56

⇒ **Ah =1, al = 200**

⇒ **Add al, 56 =>200+56 = 0**

⇒ **Ax = ah *256 + al = 1 * 256 +0= 256**

HỢP NGỮ

- **MOV** des, src

  MOV REG, memory
  MOV memory, REG
  MOV REG, REG
  MOV memory, immediate
  MOV REG, immediate

REG: AX, BX, CX, DX, AH, AL, BL, BH, CH, CL, DH, DL, DI, SI, BP, SP.
memory: [BX], [BX+SI+7], variable, etc...
immediate: 5, -24, 3Fh, 10001101b, etc...

HỢP NGỮ

MOV SREG, memory

MOV memory, SREG

MOV REG, SREG

MOV SREG, REG

SREG: DS, ES, SS, and only as second operand: CS.

REG: AX, BX, CX, DX, AH, AL, BL, BH, CH, CL, DH, DL, DI, SI, BP, SP.

memory: [BX], [BX+SI+7], variable, etc...

HỢP NGỮ

```
MOV AX, 0B800h ; set AX to hexadecimal value of
  B800h.

MOV DS, AX ; copy value of AX to DS. MOV CL, 'A'
  ; set CL to ASCII code of 'A', it is 41h.

MOV CH, 11011111b ; set CH to binary value.

MOV BX, 15Eh ; set BX to 15Eh.

MOV [BX], CX ; copy contents of CX to memory at
  B800:015E
```

HỢP NGỮ

- **Variables**

  name **DB** *value*

  name **DW** *value*

  DB - Define Byte.
  DW - Define Word.

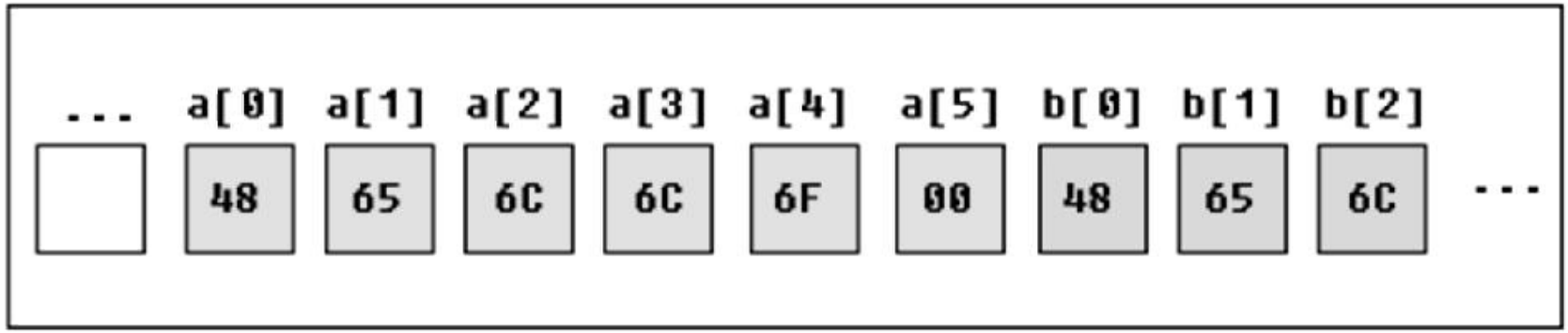
HỢP NGỮ

```
VAR1   DB   7
var2   DW   1234h
```

HỢP NGỮ

- **Arrays**

```
a DB    48h, 65h, 6Ch, 6Ch, 6Fh, 00h
b DB    'Hello', 0
```



HỢP NGỮ

- **number DUP (value(s))**

```
c   DB 5   DUP(9)
c   DB 9, 9, 9, 9, 9

d   DB 5   DUP(1, 2)
d   DB 1, 2, 1, 2, 1, 2, 1, 2, 1, 2
```

HỢP NGỮ

- **Address of a Variable**
  - **OFFSET**

    mov dx, offset s

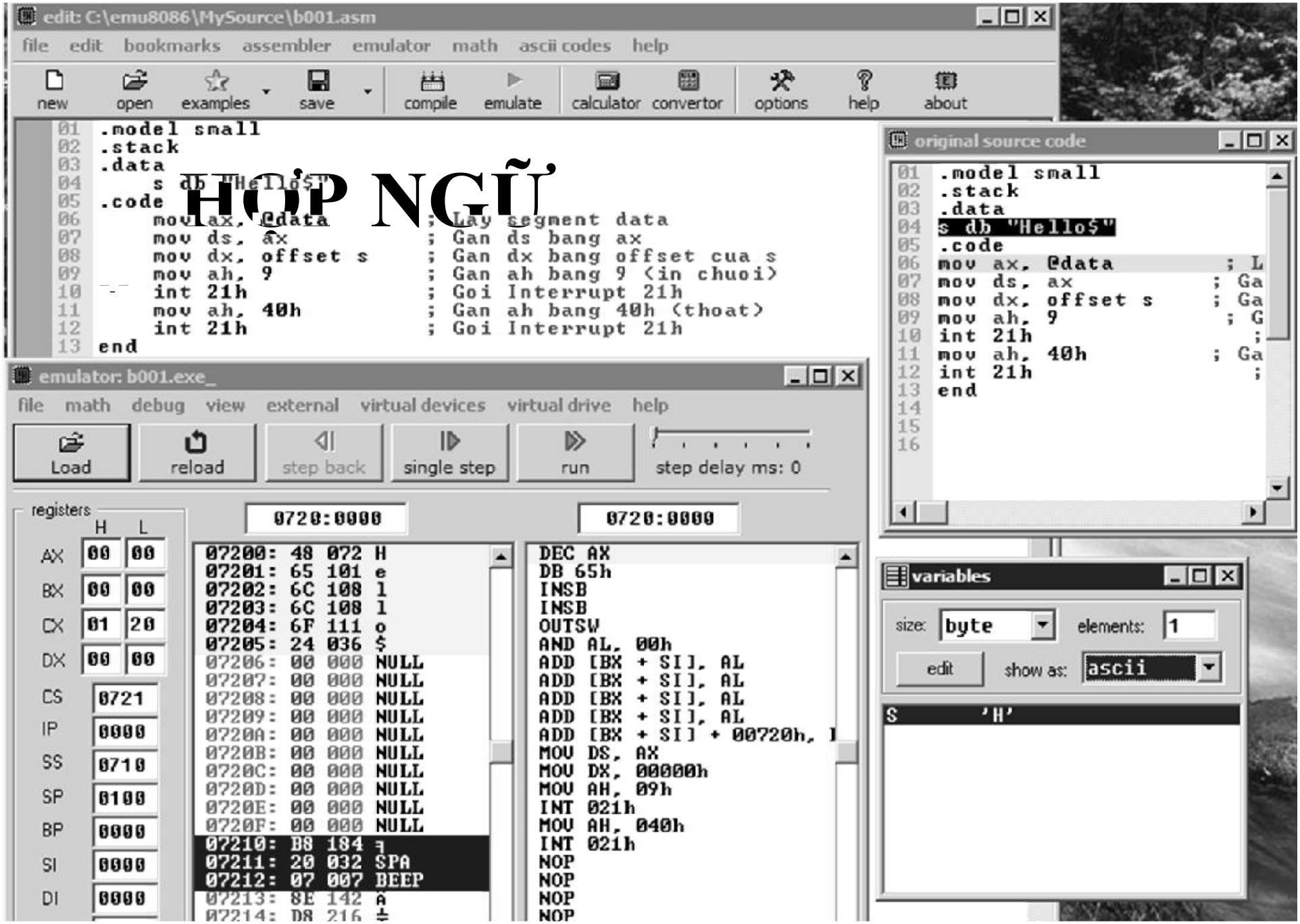
  - **LEA**

    lea dx, s

HỢP NGỮ

- **Constants**
  - name EQU expression

```
k EQU 5
MOV AX, k
```


edit: C:\emu8086\MySource\b001.asm
file edit bookmarks assembler emulator math ascii codes help
new open examples save compile emulate calculator convertor options help about
01 .model small
02 .stack
03 .data
04 s db "Hello$"
05 .code
06 mov ax, @data ; Lay segment data
07 mov ds, ax ; Gan ds bang ax
08 mov dx, offset s ; Gan dx bang offset cua s
09 mov ah, 9 ; Gan ah bang 9 (in chuoi)
10 int 21h ; Goi Interrupt 21h
11 mov ah, 40h ; Gan ah bang 40h (thoat)
12 int 21h ; Goi Interrupt 21h
13 end
HỢP NGỮ
original source code
emulator: b001.exe_
file math debug view external virtual devices virtual drive help
Load reload step back single step run step delay ms: 0
registers
H L
AX 00 00
BX 00 00
CX 01 20
DX 00 00
CS 0721
IP 0000
SS 0710
SP 0100
BP 0000
SI 0000
DI 0000
0720:0000
07200: 48 072 H
07201: 65 101 e
07202: 6C 108 l
07203: 6C 108 l
07204: 6F 111 o
07205: 24 036 $
07206: 00 000 NULL
07207: 00 000 NULL
07208: 00 000 NULL
07209: 00 000 NULL
0720A: 00 000 NULL
0720B: 00 000 NULL
0720C: 00 000 NULL
0720D: 00 000 NULL
0720E: 00 000 NULL
0720F: 00 000 NULL
07210: B8 184 ¸
07211: 20 032 SPA
07212: 07 007 BEEP
07213: 8E 142 Ä
07214: D8 216 ╪
0720:0000
DEC AX
DB 65h
INSB
INSB
OUTSW
AND AL, 00h
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI] + 00720h, I
MOV DS, AX
MOV DX, 00000h
MOV AH, 09h
INT 021h
MOV AH, 040h
INT 021h
NOP
NOP
NOP
NOP
variables
size: byte elements: 1
edit show as: ascii
$ 'H'

HỢP NGỮ

- **Interrupts**
  - **hardware interrupts**
  - **software interrupt**

- **INT value**
  - sub-function **AH** register

HỢP NGỮ

- **emu8086.inc** (include 'emu8086.inc')
  - **PRINT_STRING** - procedure to print a null terminated string at current cursor position, receives address of string in **DS:SI** register. To use it declare: **DEFINE_PRINT_STRING** before **END** directive.
  - **PTHIS** - procedure to print a null terminated string at current cursor position (just as PRINT_STRING), but receives address of string from Stack. The ZERO TERMINATED string should be defined just after the CALL instruction. For example:

    CALL PTHIS
    db 'Hello World!', 0
    To use it declare: **DEFINE_PTHIS** before **END** directive.
  - **GET_STRING** - procedure to get a null terminated string from a user, the received string is written to buffer at **DS:DI**, buffer size should be in **DX**. Procedure stops the input when 'Enter' is pressed. To use it declare: **DEFINE_GET_STRING** before **END** directive.

HỢP NGỮ

- **CLEAR_SCREEN** - procedure to clear the screen, (done by scrolling entire screen window), and set cursor position to top of it. To use it declare: **DEFINE_CLEAR_SCREEN** before **END** directive.

- **SCAN_NUM** - procedure that gets the multi-digit SIGNED number from the keyboard, and stores the result in **CX** register. To use it declare: **DEFINE_SCAN_NUM** before **END** directive.

- **PRINT_NUM** - procedure that prints a signed number in **AX** register. To use it declare: **DEFINE_PRINT_NUM** and **DEFINE_PRINT_NUM_UNS** before **END** directive.

- **PRINT_NUM_UNS** - procedure that prints out an unsigned number in **AX** register. To use it declare: **DEFINE_PRINT_NUM_UNS** before **END** directive.

HỢP NGỮ

include 'emu8086.inc'

Call PRINT_NUM_UNS
in gia tri trong thanh ghi AX dang thap phan

DEFINE_PRINT_NUM_UNS
END

HỢP NGỮ

Định vị địa chỉ:

- Định vị Tức thời (imm)
Dữ liệu cho lệnh là giá trị hằng (giá trị tức thời)

- Định vị trực tiếp : chỉ rõ địa chỉ cần truy xuất dữ liệu
Segment:[offset] (displacement)

Segmnet là các thanh ghi CS, DS, ES, SS

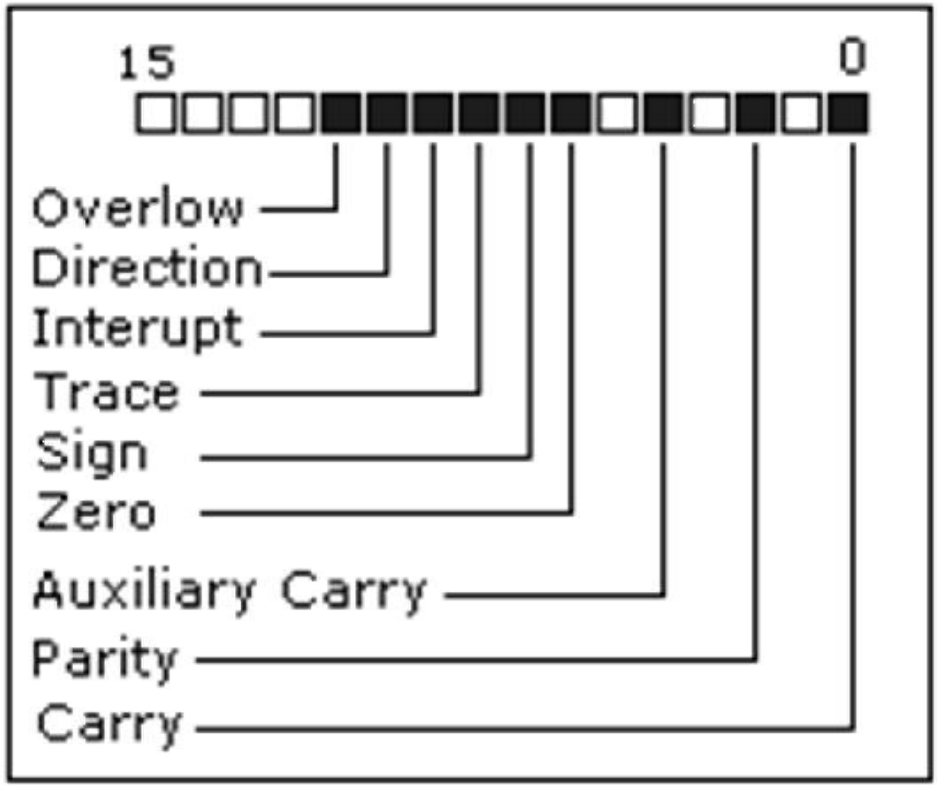
- **Định vị gián tiếp : dùng các thanh ghi để chỉ địa chỉ cần truy xuất**
- **- chỉ mục (Index): SI, DI**
- **- cơ sở (base) : BX, BP**

- Addressing Mode:

$$\begin{Bmatrix} CS: \\ DS: \\ SS: \\ ES: \end{Bmatrix} \left[\begin{Bmatrix} BX \\ BP \end{Bmatrix} \right] + \left[\begin{Bmatrix} SI \\ DI \end{Bmatrix} \right] + [\text{displacement}]$$

HỢP NGỮ

- **Flags**



HỢP NGỮ

- **Carry Flag (CF) - CF = 1 = CY khi tràn không dấu xảy ra.**
  CF = 0 = NC , không bị tràn
- **Zero Flag (ZF) - ZF = 1 = ZR khi kết quả = 0**
  ZF = 0 = NZ khi kết quả !=0.
- **Overflow Flag (OF) – OF = 1 = OV tràn có dấu xảy ra .**
  OF = 0 = NO , không bị tràn không dấu
- **Sign Flag (SF) – SF = 1 = NG khi kết quả là số âm.**
  SG = 0 = PL khi kết quả không âm.

HỢP NGỮ

- **Carry Flag (CF) - this flag is set to 1 when there is an unsigned overflow. For example when you add bytes 255 + 1 (result is not in range 0...255). When there is no overflow this flag is set to 0.**

- **Zero Flag (ZF) - set to 1 when result is zero. For none zero result this flag is set to 0.**

- **Sign Flag (SF) - set to 1 when result is negative. When result is positive it is set to 0. Actually this flag take the value of the most significant bit.**

- **Overflow Flag (OF) - set to 1 when there is a signed overflow. For example, when you add bytes 100 + 50 (result is not in range -128...127).**

HỢP NGỮ

- **Parity Flag (PF) - this flag is set to 1 when there is even number of one bits in result, and to 0 when there is odd number of one bits. Even if result is a word only 8 low bits are analyzed!**

- **Auxiliary Flag (AF) - set to 1 when there is an unsigned overflow for low nibble (4 bits).**

- **Interrupt enable Flag (IF) - when this flag is set to 1 CPU reacts to interrupts from external devices.**

- **Direction Flag (DF) - DF = 0 - xủ lý theo chiều tăng địa chỉ**

DF =1 - xủ lý theo chiều giảm địa chỉ.