Laboratory 01
MS Debug

CO 2103 Assembly Language

Objective

Using MS DEBUG to write simple AL program

- the tutorial

- data movement, writing text (char/string) to screen, reading character from keyboard (INT 21)

Microsoft® DEBUG

- Microsoft® DEBUG is the most native software debugging tool readily available free in Microsoft OS
- DEBUG is a program testing and editing tool, working at low-level:
  - checking registers' content
  - checking memory content
  - writing and testing assembly language program
  - simple IO accesses, e.g. read/write disk files, IO ports
- In short, you can write and test executable program using DEBUG, in Machine or Assembly Language

DEBUG Basics - 1

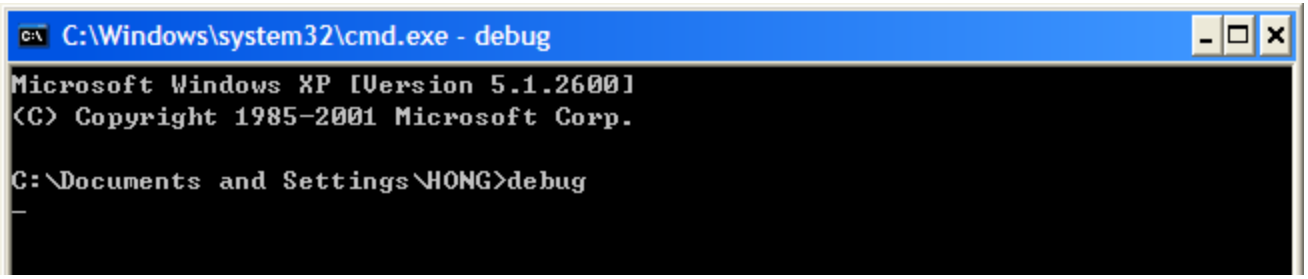
- It runs from DOS (Command) prompt
- It is command-based, with user prompt being the hyphen (-)
- All the displays and keyboard entries are in Hexadecimal
- Designed to work with .COM programs
- It can examine .EXE programs but cannot save

DEBUG Basics - 2

- When DEBUG start without filename:
  - initialize CS=DS=SS=ES
  - initialize AX, BX, CX, DX, BP, SI and DI to zero
  - initialize all flag bits to zero, except Interrupt is set to Enable
  - initialize IP=0100, SP=FFEE
- When DEBUG start with filename:
  - same as above except:
    - initialize SP=FFFE
    - CX and BX contain the size of file

Using DEBUG

- Open Command Prompt by either:
  - click Run from Start menu and enter "cmd"
  - select Command Prompt from the Accessories in All Programs of Start menu
- In the Command Prompt enter "debug"
  - the user prompt will change from ">" (DOS) to "-" (Debug)
- Enter Debug command accordingly



DEBUG Commands - 1

- Enter "?" in DEBUG will list all commands available in DEBUG

C:\Windows\system32\cmd.exe - debug

```
-?
assemble      A [address]
compare       C range address
dump          D [range]
enter         E address [list]
fill          F range list
go            G [=address] [addresses]
hex           H value1 value2
input         I port
load          L [address] [drive] [firstsector] [number]
move          M range address
name          N [pathname] [arglist]
output        O port byte
proceed       P [=address] [number]
quit          Q
register      R [register]
search        S range list
trace         T [=address] [value]
unassemble    U [range]
write         W [address] [drive] [firstsector] [number]
allocate expanded memory        XA [#pages]
deallocate expanded memory      XD [handle]
map expanded memory pages       XM [Lpage] [Ppage] [handle]
display expanded memory status  XS
-
```

DEBUG Command - 2

- Important commands:
  - **q** quit: exit DEBUG
  - **r** register: display content of registers
  - **d** dump: display content of memory locations
  - **e** enter: write into memory locations
  - **a** assemble: write assembly language program
  - **u** unassemble: decode machine codes into assembly language
  - **g** go: run the program
  - **t** trace: execute the program step-by-step (per instruction)
  - **l** load: load file (program) from disk
  - **w** write: save file (program) to disk
  - **n** name: name the file for write/load commands

DEBUG Commands - 3

- Using DEBUG commands:
  - enter only the first letter
  - not case sensitive
  - space as separator
  - separators are not usually needed, except between parameters, e.g. "–d 100 110" or "–d100 110" are both acceptable

DEBUG Commands - 4

- R (Register) - shows the status of the processor
  - display/change content of registers
    - at start, all segment registers have the same value, which points to the program segment (just above the memory space DEBUG itself use) and all other registers are cleared except: SP=FFEE (near the top of the program since stack grows downward in memory), IP=0100
    - when loading program files (.COM or .EXE) into DEBUG, SP=FFFE
    - e.g. -r ax shows the current value of AX and prompt ":" for new value
      - press enter to terminate command without changing the content
  - display/change status of flags
    - NV UP DI PL NZ NA PO NC symbolize 0 values, or clear states, of the processor flags Overflow, Direction, Disable Interrupt, Sign, Zero, Auxiliary Carry, Parity and Carry; the opposite states, set or 1, are OV DN EI NG ZR AC PE CY
    - to change the state, execute the command –rf
      - show the current states, followed by a hyphen "-", enter any number of the abbreviations after the hyphen, and the flags will be so set/cleared
  - flags will take these values only when execution of your program starts, like all the other information in the register display

DEBUG Commands - 5

- D (dump) - display bytes stored in memory
  - e.g. -d0:400 will display 128 bytes, those in addresses 0:400 to 0:47F
  - arranged in neat table with addresses on the left in segment:offset form
  - 16 bytes on a line, in two groups of 8 separated by a hyphen
  - interpreted as ASCII characters at the right, which are usually garbage
  - subsequent execution of -d, without any parameters, get the next 128 bytes
  - can see any desired number of bytes by putting L (or l) and the number of bytes at the end, e.g. -d0:400 L10 will display only 16 bytes
- E (enter) - change the bytes stored in memory
  - e.g. -e200 11 22 will store 11h into memory location ds:200 and 22h into ds:201h
  - e.g. –e200 “test” will store ASCII codes of the characters (one byte each) in memory starting from ds:200h
  - e.g. –e200 will display the byte in [ds:200h] followed by a period
    - enter new byte to change it
    - press space bar to go to the next byte
    - press enter to leave the command
    - a hyphen ("-") goes back one address

DEBUG Commands - 6

- U (Un/Disassemble) - takes the given bytes and interpret them as instructions
  - with a program, the result will be meaningful
  - for random data, it will display garbage that is meaningless
  - e.g. u100 200 disassembles bytes from memory locations starting from [cs:100h] to [cs:200h]
  - entering u without address will start from memory location [cs:0100h] or continue from where previous u command left
- A (Assemble) – assembles instructions into machine codes
  - e.g. -a100 will display the starting address [100h] for the program and wait for user to enter instructions
  - enter without typing any instruction will terminate the command
  - entering a without address will start from memory location [cs:0100h] or continue from where previous a command left

DEBUG Tutorial

- A good online resource is "A Guide to DEBUG" by Daniel B. Sedory at: http://mirror.href.com/thestarman/asm/debug/debug.htm
- **Task 1:** Go through the above guide, including the tutorial in its Page 2
  - check out the initial values of registers, flags and memory
  - know how to write, test and save a simple print screen program

Flags in DEBUG

```
Textbook abbrev. for Flag Name =>   of df if sf zf af pf cf
If the FLAGS were all SET (1),      -- -- -- -- -- -- -- --
they would look like this...   =>   OV DN EI NG ZR AC PE CY
If the FLAGS were all CLEARed (0),
they would look like this...   =>   NV UP DI PL NZ NA PO NC
```

FLAGS			SET (a 1-bit)	CLEARed (a 0-bit)
Overflow	of	=	OV	NV [No Overflow]
Direction	df	=	DN (decrement)	UP (increment)
Interrupt	if	=	EI (enabled)	DI (disabled)
Sign	sf	=	NG (negative)	PL (positive)
Zero	zf	=	ZR [zero]	NZ [Not zero]
Auxiliary Carry	af	=	AC	NA [No AC]
Parity	pf	=	PE (even)	PO (odd)
Carry	cf	=	CY [Carry]	NC [No Carry]

The first program in the tutorial – an explanation

1 start assemble program at memory location CS:0100

2 CPU will start execute from here [CS:0100] BUT

3 the actual program only start from here [CS:0126] BECAUSE

5 for that reason, we want the CPU to skip the data [from CS:0102 to 0125] and continue to execute from CS:0126

```
-a 100
xxxx:0100 jmp 126          ; Jump over data that follows:
xxxx:0102 db 0d,0a,"This is my first DEBUG program!"
xxxx:0123 db 0d,0a,"$"
xxxx:0126 mov ah,9         ; Function 09 of Int 21h:
xxxx:0128 mov dx,102       ; DS:DX -> $-terminated string.
xxxx:012B int 21           ; Write String to STD Output.
xxxx:012D mov ah,0         ; Function 00 of Int 21h:
xxxx:012F int 21           ; Terminate Program.
xxxx:0131
-g =100
```

4 these few locations [from CS:0102 to 0125] have been used to store some data, using "db"

6 these 3 instructions set up and call the DOS subroutine (software interrupt routine) to print the message starting at location 0102 (and end with '$') to the screen

7 these 2 instructions set up and call the DOS subroutine (software interrupt routine) to terminate the program

8 running the program from location CS:0100
note that same output will be obtained by "g =126"

9 output from the program

```
This is my first DEBUG program!

Program terminated normally
-
```

Exercise

- **Task 2:** Referring to the tutorial on DEBUG, write and save a hello.COM program to display the following message on your console screen:

  Hello World !

- **Task 3:** Debug the hello.com, check the relevant memory content and modify the massage to:

  Hurray, I cracked the code !!!

  and save the file as hello2.COM