Laboratory 02
More on MS Debug

CO 2103 Assembly Language

Objective

Use MS DEBUG to write AL program

-tracing and more exercises

-familiarize with common 8086 instructions

scripting in DEBUG

Tracing Program

- One of the important functions of a debugger is the ability to trace a program, i.e. monitor its execution
  - program tracing is basically executing a program step-by-step (instruction-by-instruction or block-by-block) and monitoring the outcome (changes in registers, memory, etc)
  - in this way, one can find the error (or bug) in the program by noting unexpected change in status
  - related command in Debug:
    - Go: g [=address] [addresses]
    - Proceed: p [=address] [number]
    - Trace: t [=address] [number]

Go

- Go: g [=address] [addresses]
  - run the program starting from address specified in [=address]
  - [addresses] allows the user to set up to 10 breakpoints during the execution
  - a breakpoint is an address where the program will halt, before executing the instruction at this point
    - press g again at a breakpoint will continue to run the program from this point to the end or next breakpoint
  - a breakpoint can only be set at an address containing the first byte of a valid 8088/8086 op code

Trace

- Trace: t [=address] [number]
  - step through CPU instructions one at a time, i.e. instruction-by-instruction execution
  - display CPU status (registers, flags) at each step
  - [=address] set CS:IP=address and execute the instruction (only one) at this address and then set IP=address+1
  - t without [=address] will execute the instruction pointed by IP, and then set IP=IP+1
  - can be asked to step through a number of instructions specified by [number]

Proceed

- Proceed: p [=address] [number]
  - same as Trace except:
    - immediately execute ALL the instructions (rather than stepping through each one) inside any Subroutine CALL, a LOOP, a REPeated string instruction or any software INTerrupts
    - more useful than Trace
    - Trace only be used to step into a Subroutine or possibly check the logic of the first few iterations of a LOOP or REP instruction

Simple Example - g, p, t

- mov ax, 5 ;ax=5h
- add ax, 10 ;ax=5+10=15h
- add ax, 20 ;ax=15+20=35h
- mov [0120],ax ;[0120]=ax=35h
- int 20 ;exit

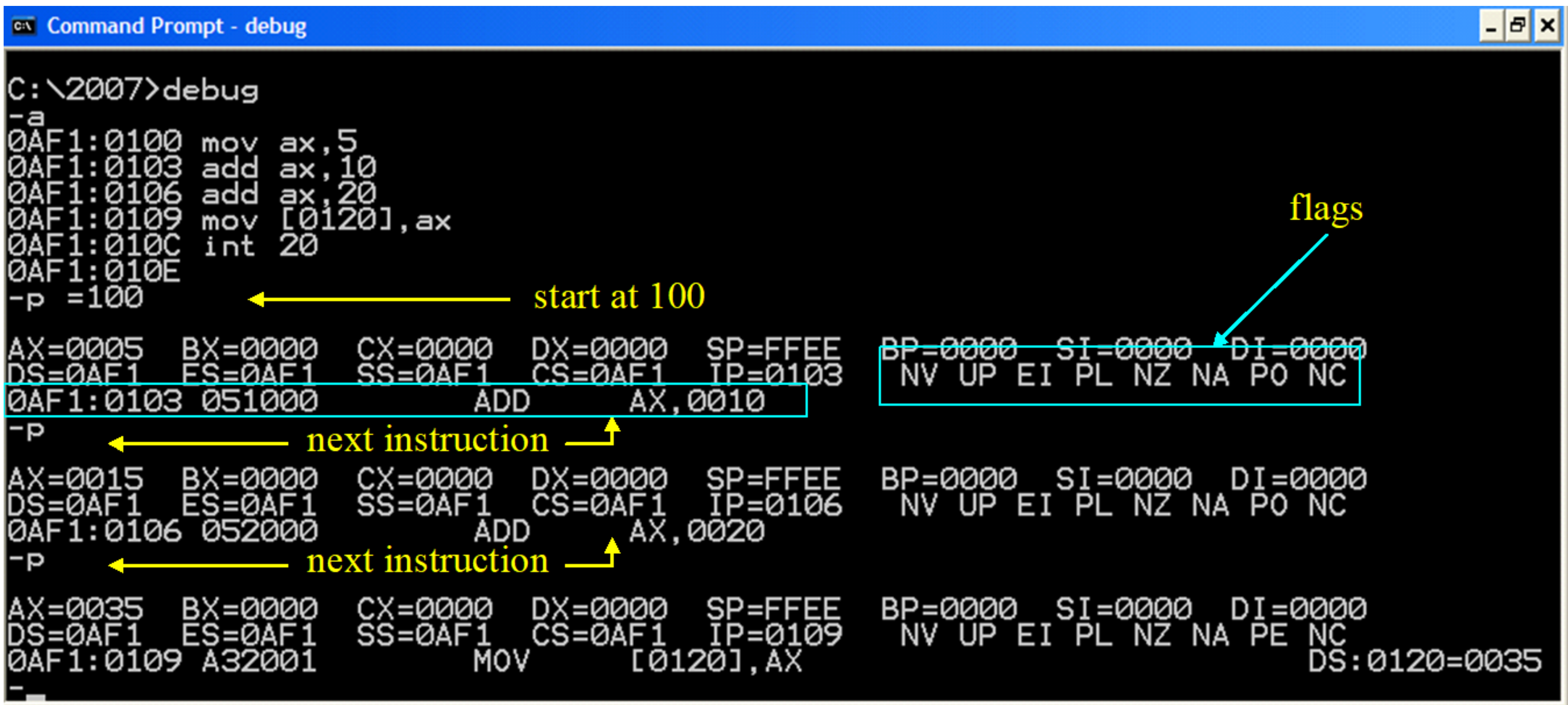
- **Task 1:** Use DEBUG to enter the above program and use **g**, **p** and lastly **t** to execute it.

Simple Example - observations

- Go: only final result is seen, i.e. you can see the final result in memory [0120] by doing a dump
- Proceed: the status of registers and flags can be observed after each step, and you can see the memory content by doing a dump in each step
- Trace: similar as Proceed, however, it steps into the INT 20 interrupt routine, which may not be of interest for program to debug (INT are assumed bug free)

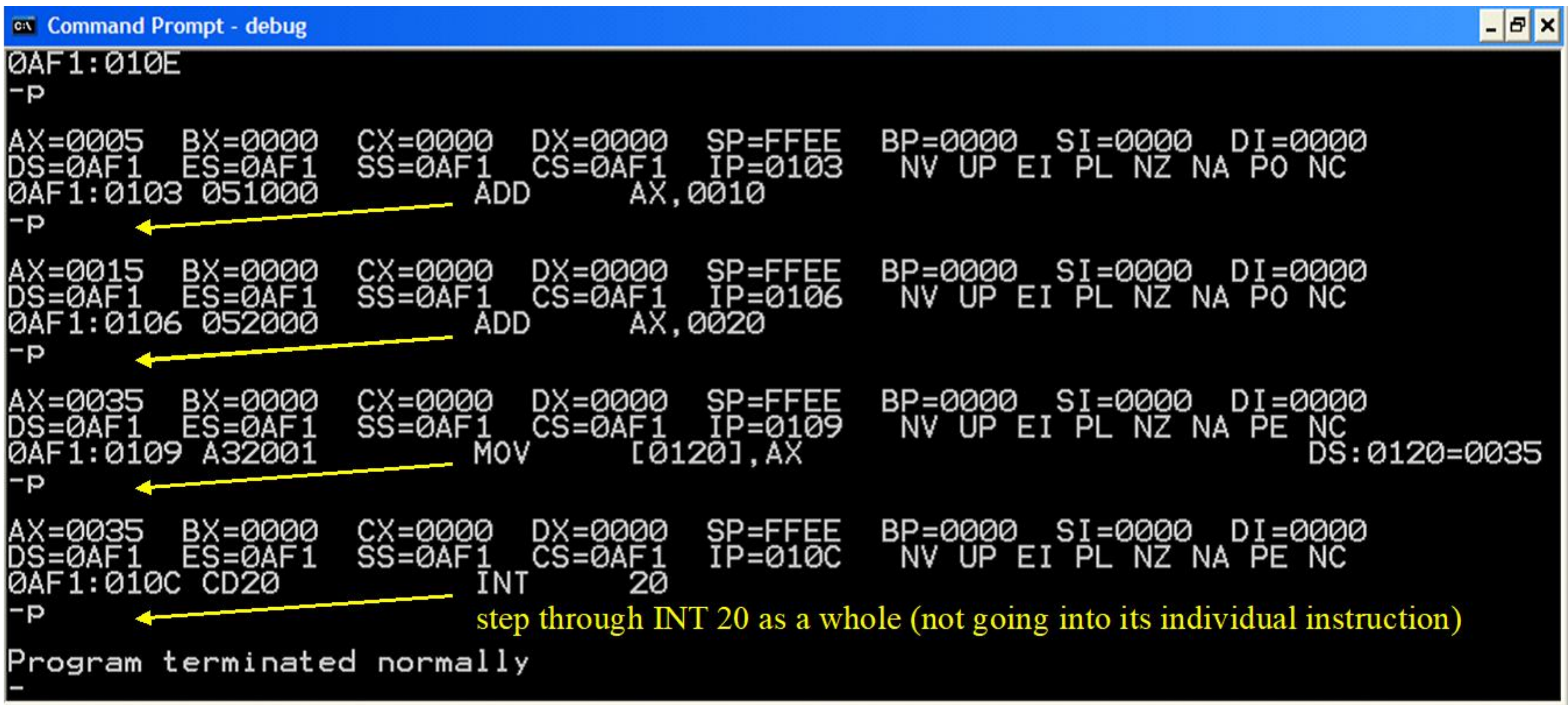
Simple Example – screenshots 1

- Proceed …



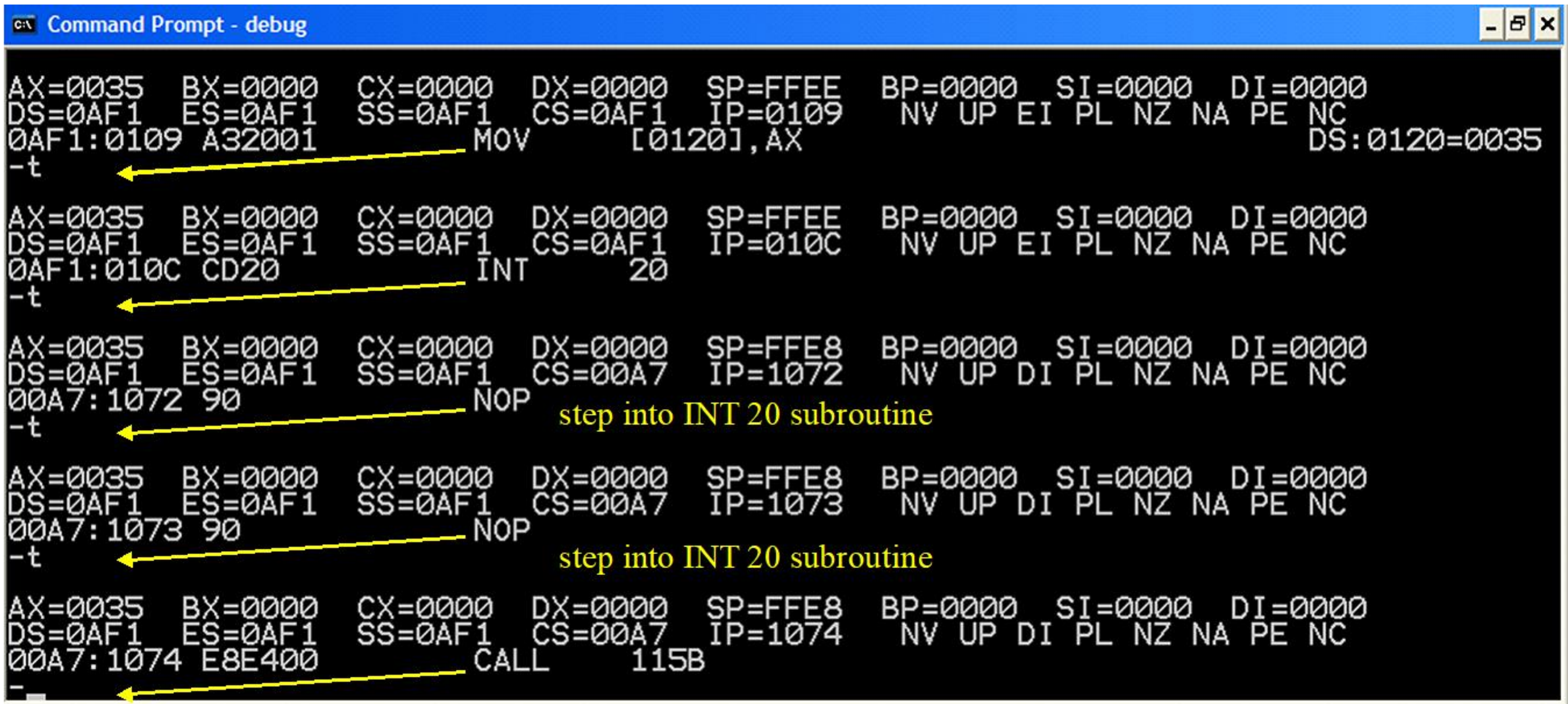
Simple Example – screenshots 2

- … Proceed



Simple Example – screenshots 3

- Trace



Exercises – common 8086 instructions

- Programming 8086 in AL can be easy as there are limited instructions to know – we usually use a small set of commonly used instructions
- It is unnecessary to remember the instructions, however, one should know how to determine the function of an instruction by referring to an instruction set
- Refer to the Complete 8086 Instruction Set at http://www.emu8086.com/assembly_language_tutorial_assembler_reference/8086_instruction_set.html (link from moodle) and do the following exercises
  - first, try to predict the result by only referring to the instruction set without using any tool
  - then, verify your answer using DEBUG

Exercises

For the following exercises, use DEBUG to assemble the whole set of instructions for each task and trace/proceed through the program

While tracing, you may need to check memory content using dump command

Exercises – data transfer

- **Task 2:** Data transfer instructions –referring to instruction set, confirm which of the following instructions (in sequence) are valid and predict the result of each of them in turn, then check you answer in DEBUG (recap: 8086 uses Little Endian scheme; hint: enter all in assembler and trace the program)
  - mov ax,0abc
  - mov bl,0abc
  - mov ch,bc
  - mov bc,ch
  - mov [200],ch
  - mov [201],ax
  - mov dl,[200]
  - mov bx,[201]
  - mov [203],34
  - mov 3456,[206]
  - xchg cl,bh
  - xchg cx,ax
  - xchg dl,cx
  - lea di,[201]

  all numbers in hex

Exercises –logic

- **Task 3:** Logic instructions – predict the result of the following instructions (in sequence) and check your answer in DEBUG (check the flags)
  - not dx
  - not [200]
  - and dl,c3
  - and dl,f0
  - or bl,b3
  - mov [200],dl
  - or bl,[200]
  - mov [201],bx
  - or bx,dx
  - or ax,5511
  - and [200],ah
  - and [201],fe
  - or [201],55
  - xor ax,ax
  - xchg dh,dl
  - xor bx,dx
  - test bx,80a1

  all numbers in hex

Exercise – arithmetic

- **Task 4:** Arithmetic instructions – predict the result of the following instructions (in sequence) and check your prediction in DEBUG (check the flags)

  - add al,75
  - add bl,89
  - add al,bl
  - inc cl
  - add al,cl
  - add bl,77
  - adc al,bl
  - sub [200],cl
  - neg cx
  - adc ax,[200]
  - sub ax,5
  - add dh,4
  - mul dh
  - add bx,20
  - div dh
  - imul dx
  - div [200]
  - dec dx
  - dec [200]
  - cmp ax,bx

  all numbers in hex

Exercises - bit manipulation

- **Task 5:** Bit manipulation instructions – predict the result of the following instructions (in sequence) and check your prediction in DEBUG (check the flags)
  - mov ax,aaaa
  - mov cl,03
  - shl ax,1
  - shl ax,cl
  - mov ax,aaaa
  - sal ax,1
  - sal ax,cl
  - mov dx,aaaa
  - shr dx,2
  - shr dx,cl
  - mov dx,aaaa
  - sar dx,2
  - sar dx,cl
  - mov bl,08
  - stc
  - rol bl,cl
  - mov bl,08
  - stc
  - rcl bl,cl
  - stc
  - ror bl,cl

all numbers in hex

Scripting in DEBUG

- Inconvenient to write program in DEBUG
- Alternative: use DEBUG script (.scp file)
  - write DEBUG commands in text file
  - load into DEBUG
    - debug < file.scp
- **Task 6:** Try out this tutorial (DEBUG only): http://thestarman.pcministry.com/asm/fire/Fire.html
  - note the ability of AL in control the hardware, in this case the screen