Basics of 8086 Assembly Language Programming

CO 2103 Assembly Language

Topics

- Tools in AL Programming
- AL Programming Process
- “Hello World!” in AL
- Skeleton in brief
- Elements of AL Programs
  - Statement
  - Label
  - Directive / Pseudo-Opcode
- Software Interrupt (Function Call)

Assembly vs Machine Language
(recap)

- Machine Language (Machine Codes) – what CPU understands, in 0s and 1s only, BUT not readable to human
- Assembly Language (Assembly Codes) – representation of ML in symbolic form, for improved readability to human, BUT not readable to CPU
  - uses some high-level (HL) notations
- Require translation from AL to ML before saving into memory for CPU to process/execute

Tools in AL Programming - 1

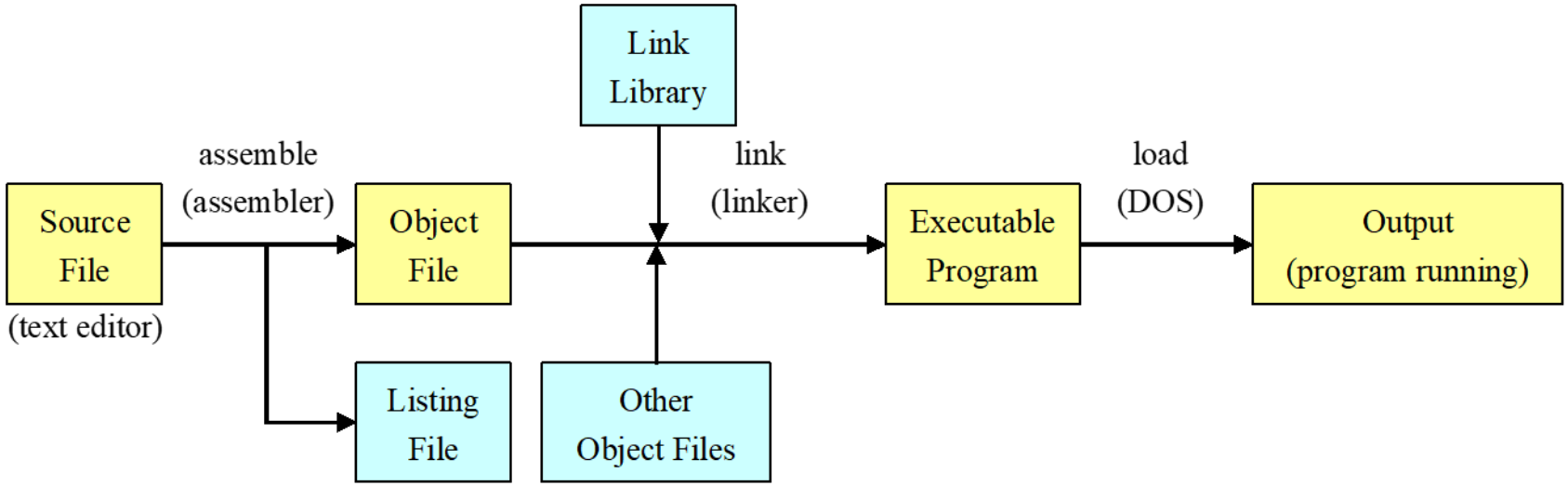
- Assembler – convert AL program into Machine Code (ML)
  - functions:
    - fix memory location for symbolic addresses
    - change mnemonics to machine codes
    - resolve symbols into actual data
    - generate information for the Linker including:
      - entry point (i.e. where program execution should start)
      - addresses of symbols declared in this file (which may be used in other files)
      - unresolved symbols (which may be defined in other files)
  - can be regarded as a low-level compiler
  - output is Object file (.obj): same format as a compiler output
  - can also output Listing file (.lst): lists program in AL and ML

Tools in AL Programming - 2

- Linker – combine one or more object files into an executable file
  - functions:
    - merge fragments of data and codes from different object files
    - may change the offset of some of the symbols
    - adjust all references to these symbols
    - generate information for relocation
  - may also extract object codes from a library file
  - output is executable file (.exe)
- Text Editor (ASCII) – to create the AL source file

AL Programming Process

- Edit source file – AL → .asm
- Assemble AL source file: AL into ML → .obj and .lst
- Link ML Object file(s): address relocation → .exe
- Load (run) executable: into available memory space



AL Source File – hello.asm

```
① ; This program displays "Hello, world!"
② .model small
③ .stack 100h

④ .data
⑤ message db "Hello, world!",0dh,0ah,'$'   ;newline + eoc

⑥ .code
⑦ main: mov ax,@data                       ; data segment
        mov ds,ax

        mov ah,9
        mov dx,offset message
        int 21h                            ; display msg starting at 0

        mov ax,4c00h                       ; halt the program and exit
        int 21h

⑧ end main
```

Elements of AL Source File - 1

- Constants and expressions
  - numeric literals
  - character or string constant
- AL directives
  - memory defining directives
  - model directive: .model
  - segment directives: .data, .stack, .code
  - program organising directives - title, proc, endp, end
- Comment statement begins with semicolon ";"
- AL **Statement**:
  - first column may be optional label – reference to address or data
  - second column may be op code or assembler directive
  - last column may be comment beginning with semicolon ";"
  - AL Statement format:

```
<label>     <op code or directive>  <operands>              ;<comments>
message     db          "Hello, world!",0dh,0ah,'$'                 ;newline + eoc
main:       mov         ax,@data                                    ; data segment
```

Elements of AL Source File - 2

- A numeric literal is any combination of digits, plus optional decimal point, exponent, sign
  - use a radix symbol (suffix) to select binary, octal, decimal, or hexadecimal

```
5234                6A15h       ; hexadecimal
5.5                 0BAF1h      ; leading zero required
-5.5                32q         ; octal
26.0E+05            1011b       ; binary
+35d                35d         ; decimal (default)
```

- Symbolic constants are defined using the EQU directive or '=' operator
  - must evaluate to a 16-bit integer
  - or 32-bit integer when .386 directive used

```
COUNT EQU 25
ROWS = 10
tablePos = ROWS * 5
```

Elements of AL Source File - 3

- Symbolic constants can be defined with constant expression - combination of numeric literals, operators, and defined symbolic constants
  - must be evaluated at assembly time

```
SIZE = 4 * 20
NUM = -3 * 4 / 6
NROW = ROWS - 3          ;ROWS is a constant
REM = COUNT MOD 5
```

hello.asm explained - 1

- A comment stating the program function; you can also start with TITLE directive to give a program title
- Assembler directive to reserve stack (size) in memory
- Assembler directive (Assembler dependent) to declare the memory model used. Model information tells the linker how to merge the various data and code segments.
  - Tiny data+code in one segment ≤64K (.com)
  - Small data in one segment, code in one segment
  - Compact data in multiple segments, code in one segment
  - Medium data in one segment, code in multiple segments
  - Large data and code in multiple segments
  - Huge allow data segment to be larger than 64K
  - Flat no segment, 32-bit addresses, protected mode only (80386 and higher)

hello.asm explained - 2

- Assembler directive that tells the assembler to assemble the following instructions into a data segment.
  - where the data are defined: good to place this after the code
- "message" is the label for the first memory address in data segment – "message" will be assembled and linked to a memory address
  - db is data defining directives to define bytes to be stored at locations starting at "message"
  - the statement stores 16 bytes (1 byte of ASCII code for each character) into the data segment
  - other data defining directives: dw, dd, dq, dt (more on these later)

hello.asm explained - 3

- Assembler directive that tells the assembler the following instructions into a code segment
  - where the program is written
- Start (entry) of the program instructions:
  - first two instructions initialize DS register to point to correct Data Segment - @data is an immediate operand, which actual value will be patched in by the OS during loading
  - next three instructions set up and call the DOS function (software interrupt routine) to display the characters starting at location labeled with message, and ending with "$"
  - last two instructions set up and call the DOS function to halt the program and exit
- Assembler directive that marks the end of AL program

Skeleton in brief

```
.model small/medium/...          ;specify memory model
.stack size                      ;specify stack size
.data                            ;data segment
<data declaration>               ;declare data (variables, etc)

.code                            ;code segment
proc1 PROC (near or far)         ;declare procedure proc1
        <statements>             ;codes here – end with return instruction
proc1 endp                       ;end of procedure proc1

proc2 PROC (near or far)         ;declare procedure proc2
        <statements>             ;codes here – end with return instruction
proc2 endp                       ;end of procedure proc2

main PROC                        ;begin of main program
        <statements>             ;codes here
main endp                        ;end of main program

end                              ;end of AL program
```

Label

- Label (or Symbol or Identifier) is a name associated with some particular value :
  - memory address
  - data (constant or variable)
- Analogy to variables in mathematical expressions, label are used as variables in the program
- Provides the ability to represent some otherwise incomprehensible value with a familiar, mnemonic, name.
- In hello.asm:
  - message, main are labels

Rules for Label

- Label consists of a sequence of letters, digits, and special characters, with the following restrictions:
  - cannot begin with a numeric digit
  - usually not case sensitive
  - may contain any number of characters, however only the first 31 are used
  - _, $, ?, and @ symbols may appear anywhere within a label
  - $ and ? are special symbols; you cannot create a label made up solely of these two characters.
  - cannot match any name that is a reserved symbol

Reserved words

- Apart from all valid 80x86 instruction names and register names, words in the list on the right are reserved, i.e. cannot be used as label:

```
%out        .186        .286        .286P
.287        .386        .386P       .387
.486        .486P       .8086       .8087
.ALPHA      .BREAK      .CODE       .CONST
.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      .MSFLOAT    .NO87       .NOCREF
.NOLIST     .NOLISTIF   .NOLISTMACRO .RADIX
.REPEAT     .UNTIL      .SALL       .SEQ
.SFCOND     .STACK      .STARTUP    .TFCOND
.UNTIL      .UNTILCXZ   .WHILE      .XALL
.XCREF      .XLIST      ALIGN       ASSUME
BYTE        CATSTR      COMM        COMMENT
DB          DD          DF          DOSSEG
DQ          DT          DW          DWORD
ECHO        ELSE        ELSEIF      ELSEIF1
ELSEIF2     ELSEIFB     ELSEIFDEF   ELSEIFDEF
ELSEIFE     ELSEIFIDN   ELSEIFNB    ELSEIFNDEF
END         ENDIF       ENDM        ENDP
ENDS        EQU         EVEN        EXITM
EXTERN      EXTRN       EXTERNDEF   FOR
FORC        FWORD       GOTO        GROUP
IF          IF1         IF2         IFB
IFDEF       IFDIF       IFDIFI      IFE
IFIDN       IFIDNI      IFNB        IFNDEF
INCLUDE     INCLUDELIB  INSTR       INVOKE
IRP         IRPC        LABEL       LOCAL
MACRO       NAME        OPTION      ORG
PAGE        POPCONTEXT  PROC        PROTO
PUBLIC      PURGE       PUSHCONTEXT QWORD
REAL4       REAL8       REAL10      RECORD
REPEAT      REPT        SBYTE       SDWORD
SEGMENT     SIZESTR     STRUC       STRUCT
SUBSTR      SUBTITLE    SUBTTL      SWORD
TBYTE       TEXTEQU     TITLE       TYPEDEF
UNION       WHILE       WORD
```

Directive / Pseudo-Opcode

- Directives are special instructions that provide information to the assembler but do not generate any code, e.g. segment directives, equ, assume, end. They are not valid 80x86 instructions. They are messages to the assembler, nothing else.
- Pseudo-Opcode is a message to the assembler, just like an assembler directive. They are sometimes used interchangeably . However, a pseudo-opcode will emit object code bytes, e.g db, n dup(?). These instructions emit the bytes of data specified by their operands but they are not true 80x86 machine instructions.

Data Defining Directives - 1

Mnemonic	Description	Bytes	Operand
DB	Define Byte	1	Byte
DW	Define Word	2	Word
DD	Define Doubleword	4	Doubleword
DF, DP	Define Far Pointer	6	Far Pointer
DQ	Define Quadword	8	Quadword
DT	Define Tenbytes	10	Tenbyte

- Data types can be decimal (100), binary (100b), hexadecimal (100h) or ASCII ('100' or "100")
  - no typing – up to the programmer to define type
  - use radix (suffix) to select binary, octal, decimal or hexadecimal

Data Defining Directives - 2

```
.data
char1       db    'A'
signed1     db    -128,100
combo       db    'A',-10,30h,1101b,?,24
list        db    10,20,30,40
aString     db    "Hello there",0
count       db    ?                        ;uninitialised
wordVal     dw    1234h,65535,-24
largeVal    dd    12345678h
```

- Use comma to define a list of values
  - can mix different representations (decimal, binary, hex, ASCII)
- "?" represents an uninitialised memory location (allocated)
- Word, doubleword and quadword data are stored in reverse byte order (in memory)

Directive	**Bytes in memory**
db 1234567h	67 45 23 01

Data Defining Directives - 3

- DUP
  - allows a sequence of storage locations to be defined (with same value) or reserved (uninitialised)

```
db  40 DUP (?)       ;reserve 40 bytes storage
db  30 DUP (10h)     ;allocate 30 bytes storage, each initialized to 10h
```

- Equal sign "="

Label = expression

  - expression must be numeric
  - can be redefined in program

```
count = 1
count = count * 2
```

- EQU

Label EQU expression

  - expression can be string or numeric
  - use < and > to specify a string
  - cannot be redefined in the program

```
val1        EQU     7Fh
message     EQU     <This is a message>
```

DOS Function Calls – IO

- AL can make use of software interrupt (function call) to access the IO (keyboard and screen)
  - these are useful predefined subroutine
- Called using INT instruction in AL program

  INT n ;software interrupt number n
  - INT 10h - video BIOS
  - INT 14h - serial I/O
  - INT 16h - keyboard BIOS
  - INT 17h - printer services
  - INT 1Ah - time of day
  - INT 1Ch - user timer
  - INT 21h - DOS services

DOS Function Call – few e.g.

--------D-2100--------------------------------
INT 21 - DOS 1+ - TERMINATE PROGRAM
AH = 00h
CS = PSP segment
Notes: Microsoft recommends using INT 21/AH=4Ch for DOS 2+
execution continues at the address stored in INT 22 after DOS performs whatever cleanup it needs to do
if the PSP is its own parent, the process's memory is not freed; if INT 22 additionally points into the terminating program, the process is effectively NOT terminated
not supported by MS Windows 3.0 DOSX.EXE DOS extender
SeeAlso: AH=26h,AH=31h,AH=4Ch,INT 20,INT 22

--------D-2101--------------------------------
INT 21 - DOS 1+ - READ CHARACTER FROM STANDARD INPUT, WITH ECHO
AH = 01h
Return: AL = character read
Notes: ^C/^Break are checked, and INT 23 executed if read
character is echoed to standard output
standard input is always the keyboard and standard output the screen under DOS 1.x, but they may be redirected under DOS 2+
SeeAlso: AH=06h,AH=07h,AH=08h,AH=0Ah

--------D-2102--------------------------------
INT 21 - DOS 1+ - WRITE CHARACTER TO STANDARD OUTPUT
AH = 02h
DL = character to write
Return: AL = last character output (despite the official docs which state nothing is returned) (at least DOS 3.3-5.0)
Notes: ^C/^Break are checked, and INT 23 executed if pressed
standard output is always the screen under DOS 1.x, but may be redirected under DOS 2+
the last character output will be the character in DL unless DL=09h on entry, in which case AL=20h as tabs are expanded to blanks
SeeAlso: AH=06h,AH=09h

--------D-2107--------------------------------
INT 21 - DOS 1+ - DIRECT CHARACTER INPUT, WITHOUT ECHO
AH = 07h
Return: AL = character read from standard input
Notes: does not check ^C/^Break
standard input is always the keyboard under DOS 1.x, but may be redirected under DOS 2+
if the interim console flag is set (see AX=6301h), partially-formed double-byte characters may be returned
SeeAlso: AH=01h,AH=06h,AH=08h,AH=0Ah

--------D-2108--------------------------------
INT 21 - DOS 1+ - CHARACTER INPUT WITHOUT ECHO
AH = 08h
Return: AL = character read from standard input
Notes: ^C/^Break are checked, and INT 23 executed if detected
standard input is always the keyboard under DOS 1.x, but may be redirected under DOS 2+
if the interim console flag is set (see AX=6301h), partially-formed double-byte characters may be returned
SeeAlso: AH=01h,AH=06h,AH=07h,AH=0Ah,AH=64h

--------D-2109--------------------------------
INT 21 - DOS 1+ - WRITE STRING TO STANDARD OUTPUT
AH = 09h
DS:DX -> '$'-terminated string
Return: AL = 24h (the '$' terminating the string, despite official docs which state that nothing is returned) (at least DOS 3.3-5.0)
Notes: ^C/^Break are checked, and INT 23 is called if either pressed
standard output is always the screen under DOS 1.x, but may be redirected under DOS 2+
under the FlashTek X-32 DOS extender, the pointer is in DS:EDX
SeeAlso: AH=02h,AH=06h"OUTPUT"

.... and more

Must Know INT

- Note the two-line instructions to exit to DOS
  - They will probably appear at the end of most, if not all, programs you will write

```
; Minimal program: do nothing and exit to DOS
.model small
.stack 100
.data
.code
main:   mov ax,4c00h            ; halt the program and return
int 21h
end main
```

AL programming exercises … (next)

Summary

- CPU understands ML, human understands AL
  - requires translation between AL and ML
- AL Programming Process:
  - Edit → Assemble → Link → Load (Run)
- Tools for AL Programming:
  - Editor, Assembler, Linker
- Elements of AL Source File:
  - Statement: label, op code, operand, directive, comment
  - Directive: program organization, memory, data defining
- INT function calls – useful for IO