Lecture 6

Assembly Language Requirements

Text: Chapter 4

Assembly Language

Assembly language allows the programmer to write machine instructions symbolically rather than in binary.

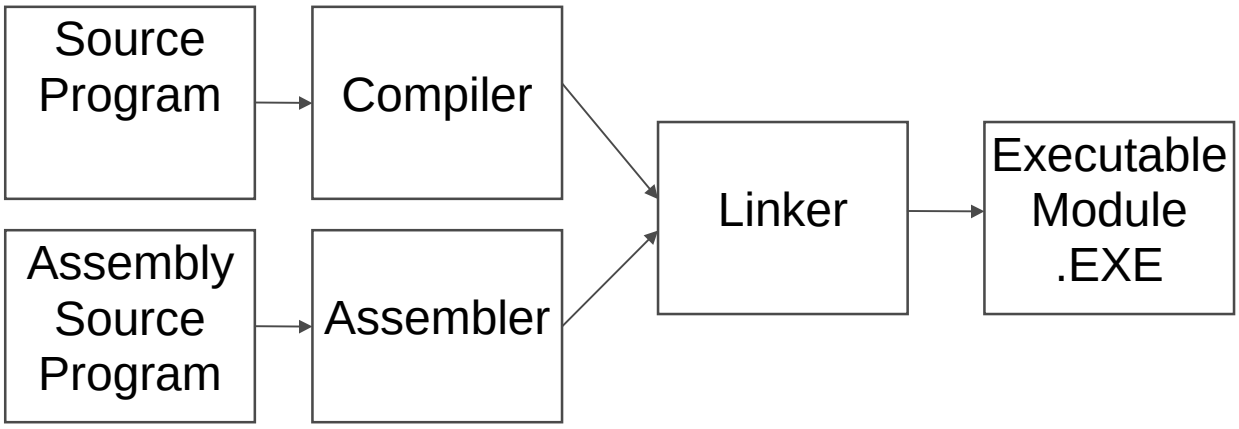
There is no need to

- remember complicated binary instruction formats
- calculate numeric offsets for variables

It is rare to need to program in assembly language because high-level language compilers generate fairly efficient code and are flexible.

Occasionally assembly language is used because it
- provides more control over hardware
- generates smaller executable modules
- generates programs that execute faster

Since all programs eventually end up in binary, there is no reason why the two can't be mixed, reserving assembly language for special needs:



Basics of Assembly Language

TASM (Turbo Assembler)
MASM (Microsoft Assembler)

Comments

Anything following a semicolon (';') is considered a comment and is ignored by the assembler.

There are three kinds of comments, and all of them are required in programs written for this course:

Abstract

A paragraph at the beginning of your program where you include your name, course, section, and a description of what the program does.

```
1. ;        Charles Babbage
2. ;        Computer Science 141  KA
3. ;
4. ;   Calculate the gross pay and
5. ;   bonus less taxes for all …
```

Section Comments

A "section" of your program is somewhat hard to define, but it is where a group of instructions solve a particular part of the problem.

```
18. ;  Calculate the gross pay by
19. ;  hours worked times rate, and
20. ;  add overtime at 1.5 the rate…
```

Line comments

Assembly language statements are terse and cryptic, and require explanation so the reader doesn't spend a lot of time wondering why you are doing something.

Every statement should have a comment describing **why** it is in the program. (Not **what** it does!)

```
25.    MOV  AL,HOURS ;get the hours
26.    MUL  RATE     ;times the rate
```

NOT:

```
25.    MOV  AL,HOURS ;put hours in AL
26.    MUL  RATE     ;multiply by RATE
```

Use white space to the reader's benefit… remember

Someone else is going to read your program!

Reserved Words

Certain words may only be used by the assembler

- instructions MOV, ADD
- directives END, SEGMENT
- operators FAR, SIZE
- functions @Data, @Model

(They are listed in Appendix C of the textbook.)

Identifiers

Variable names (to refer to data) and labels (to refer to instructions). May contain up to 31 letters, numbers and special characters (? _ $ @ .). The first letter must be alphabetic or a special character (except '.', and avoid starting with '@'). There is no distinction between upper and lower case, so CAT is the same as Cat is the same as cat.

CAT
R2D2
Gross_Salary_Minus_Taxes
$alary

Statements

There are two types of statements:

Symbolic Machine Instructions

The actual machine instructions that are translated into binary and executed by the CPU.

```
MOV     AL,RATE
```

Directives

Statements which tell the assembler to do something while the program is being translated into binary.

```
RATE    DW      8
```

This tells the assembler to Define a Word in memory, put the number 8 in it and call it RATE.

Each statement has four FIELDS

Identifier	Operation	Operands	Comment
RATE	DW	8	;$8 per hour
HOURS	DB	40	;40 hrs/wk
AGAIN	ADD	AX,RATE	;get rate
	PAGE	60,132	
	TITLE	FIRST My First Program	

Other important directives

The SEGMENT directive

Inform the assembler that a new segment is being written (for .EXE programs), how it is to be aligned, how it may be combined with other segments, and how it may be linked.

```
name    SEGMENT  align  combine 'class'

MyStack  SEGMENT  PARA   STACK   'Stack'
…
…
…
MyStack  ENDS          ;this directive ends
                       ;the segment
```

The ASSUME directive

You must tell the assembler the purpose of each segment in the program so that it will use the correct segment registers (i.e., use the CS register with your code segment).

```
    ASSUME  SS:MyStack,CS:MyCode,DS:MyData
```

The END directive

Placed at the end of the program. If the program is to be executed, its operand is the name of the main program procedure as defined with the PROC directive.

```
        page 60,132
        TITLE P04ASM1(EXE)Move and add operations
; -----------------------------------------
STACKSG SEGMENT PARA STACK 'Stack'
        DW     32 DUP(0)
STACKSG ENDS
; -----------------------------------------------
DATASG  SEGMENT PARA 'Data'
FLDA    DW     250
FLDB    DW     125
FLDC    DW     ?
DATASG  ENDS
; -------------------------------------------------
CODESG  SEGMENT PARA 'Code'
BEGIN   PROC    FAR
        ASSUME  SS:STACKSG,DS:DATASG,CS:CODESG
        MOV     AX,DATASG  ;Set address of DATASG
        MOV     DS,AX      ; in DS register

        MOV    AX,FLDA     ;Move 0250 to AX
        ADD    AX,FLDB     ;Add  0125 to AX
        MOV    FLDC,AX     ;Store sum in FLDC
        MOV    AX,4C00H    ;Exit to DOS
        INT    21H
BEGIN   ENDP               ;End of procedure
CODESG  ENDS               ;End of segment
        END    BEGIN       ;End of program
```

Example of complete .EXE program

Notice that program termination is done by an interrupt:

```
    MOV    AH,4Ch     ;termination code
    MOV    AL,0       ;return code 0
    INT    21h        ;exit to DOS
```

Simplified Segment Directives

.MODEL	how many code/data segments
.STACK [size]	define the stack [and its size]
.DATA	declare the DATA segment
.CODE [name]	declare the CODE segment

Model	# code segments	# data segments
TINY	used only for .COM programs	
SMALL	1	1
MEDIUM	more than 1	1
COMPACT	1	more than 1
LARGE	more than 1	more than 1

```
        page   60,132
TITLE   P04ASM2 (EXE)  Move and add operations
;-----------------------------------------------
        .MODEL  SMALL
        .STACK  64              ;Define stack
        .DATA                   ;Define data
FLDA    DW     250
FLDB    DW     125
FLDC    DW     ?
;-----------------------------------------------
        .CODE                   ;Define code segment
BEGIN   PROC    FAR
        MOV    AX,@data         ;Set address of DATASG
        MOV    DS,AX            ; in DS register
        MOV    AX,FLDA          ;Move 0250 to AX
        ADD    AX,FLDB          ;Add  0125 to AX
        MOV    FLDC,AX          ;Store sum in FLDC
        MOV    AX,4C00H         ;Exit to DOS
        INT    21H
BEGIN   ENDP                    ;End of procedure
        END    BEGIN            ;End of program
```

DATA Definition

```
[Name]    DIRECTIVE    expression
```

Name

a valid unique identifier.

Directive

DB	Define a byte
DW	Define a word
DD	Define a doubleword

Expression

? indicates no initial value

```
    DW      ?
```

a single value

```
    DW      25
    DB      12h
    DW      10010100b
    DB      “Queens College’s Best”
    DB      ‘Computer Science’’s finest’
```

a list of values

```
    DW      1,2,3,4,5,6,7
```

a repeated value

```
    DW      10 DUP(0)   ;defines 10 zeros
```

Exercises - Lecture 6

1. Write an .EXE file in assembly language which contains the following:

   a) A STACK segment which defines a stack of 64 bytes.
   b) A DATA segment with the following identifiers:

      "PageCount", a word with an initial value of 1.
      "Heading", a string (sequence of bytes) with the value "Computer Science 141"
      "GradeList", a sequence of 20 bytes which all contain zero.

   c) A CODE segment which

      Puts the value of "PageCount" into the AX register
      Puts the first byte of the "GradeList" into the CL register.