Development environments

Two styles – Assembler/compiler-linker – like MASM and Linux

and

Complete environments – like SPIM and Visual Studio

Separate development components

- Assembler/compiler-linker – like MASM and Linux
- Components are
  - Editor – your choice
    - MASM – use wordpad, Word (save as text), edit (if in DOS)
    - Linux – emacs – (I use vi because of tradition – you use emacs)
  - Assembler
    - MASM – is an assembler – called MASM (Microsoft assembler)
    - Linux – gcc contains an assembler – invoked when your file has a .s extension, otherwise gcc understands .c, .cc, .h, .o .a
    - Linux and visual C also have online assembly – assembly statements in a C program
  - Linker
    - MASM – it is called LINK – can link .OBJ, .LIB, makes a .EXE
    - Linux – invoked by gcc – can deal with .s, .c, .o, .a

Separate development components (cont.)

- Debugging
  - MASM – codeview and symdeb
    - Symdeb is more primitive – goes with MASM 5.10
    - Codeview goes with MASM 6.11
  - Linux – gdb – has help menu
    - Can be used with assembler as well as C – look at registers and stack
- What all debuggers do:
  - Examine modify, and use
    - Code
      - Run
      - Step
      - Set breakpoints
    - Data
      - global data
        - Examine in various formats
        - Modify
        - Set watchpoints (in some debuggers)
      - Stack
        - Examine
        - Modify
        - Trace stack frame (gdb does this)

File types and extensions

File types

File type	Dos/windows	Linux
Assembler	.ASM	
C source	.C	.c
C++ source	.CPP	.cc
object	.OBJ	.o
Library	.LIB	.a
Executable	.EXE or .COM	none

Assembler is just another programming language

- Stages of compiling any language
  - Lexical analysis
    - Separates an input into tokens
      - Numbers, operators, keywords, identifiers, quoted strings, comments
  - Syntactic analysis
    - Is this a legal (even if meaningless) program in this language
      - Constructs are
        - Expressions, control statements, compound statements, function calls and definitions
  - Semantic analysis
    - What does this program mean (in terms of the language definition)
    - Translation into intermediate code (often a sequence of triples or quadruples)
  - Code generation
    - Generates an object program for the target machine and operating system
    - Can even be assembly language – gcc can do this if asked (-S)

Assembler lexical and syntactical analysis

- Different assemblers have somewhat different lexical definitions
  - All are line-oriented
  - Number formats differ
    - 0FFFCH – masm, not SPIM or line assembler
    - 0xfffch – SPIM, line assembler, C
  - Keywords are usually detected at this stage
- Syntactical analysis
  - In HLL (higher-level language) this detects if a string of tokens is a legal program in the language
  - Assembler works line-by-line in two passes
    - Pass one
      - Scan all the lines and define all the labels
      - This means finding out the length of every instruction and data definition so every symbol's location and type are known
    - Pass two
      - Go through the input again and fill in values for all the operand fields
      - Make a relocation table so the object file can be linked and loaded
      - This pass includes semantic analysis and code generation in an assembler but not in HLL

An assembler statement (in MASM) has four fields

- Label (optional)
  - Using this label addresses this location
- Operation field (required)
  - This line codes this machine instruction
  - or, this line defines data
  - or, this is an assembler pseudoinstruction
- Operand fields (required by some operations)
  - separated by commas
  - Type identified by assembler as
    - Immediate (numeric or symbol equated to expression)
    - Register (name is that of a register)
    - Memory location (defined, definition defines type)
- Comment field (needed for readability)
  - starts with semicolon (;)

What follows is examples of assembly code

- Types
  - Inline assembler
    - This came from Visual studio help
    - You can also do inline in C
    - Look for it is /usr/src/linux/arch/i386, etc. (this is the Linux source code, examples are in several .s files for several architectures)
  - Version 6.11
    - A home-generated example

Comparison of inline and separate .asm

```
; POWER.ASM
; Compute the power of an integer
;
            PUBLIC _power2
_TEXT       SEGMENT WORD PUBLIC 'CODE'
_power2     PROC
            push ebp                ; Save EBP
            mov ebp, esp            ; Move ESP into EBP so we can refer
                                    ; to arguments on the stack
            mov eax, [ebp+4]        ; Get first argument
            mov ecx, [ebp+6]        ; Get second argument
            shl eax, cl             ; EAX = EAX * ( 2 ^ CL )
            pop ebp                 ; Restore EBP
            ret                     ; Return with sum in EAX
_power2     ENDP
_TEXT       ENDS
END
```

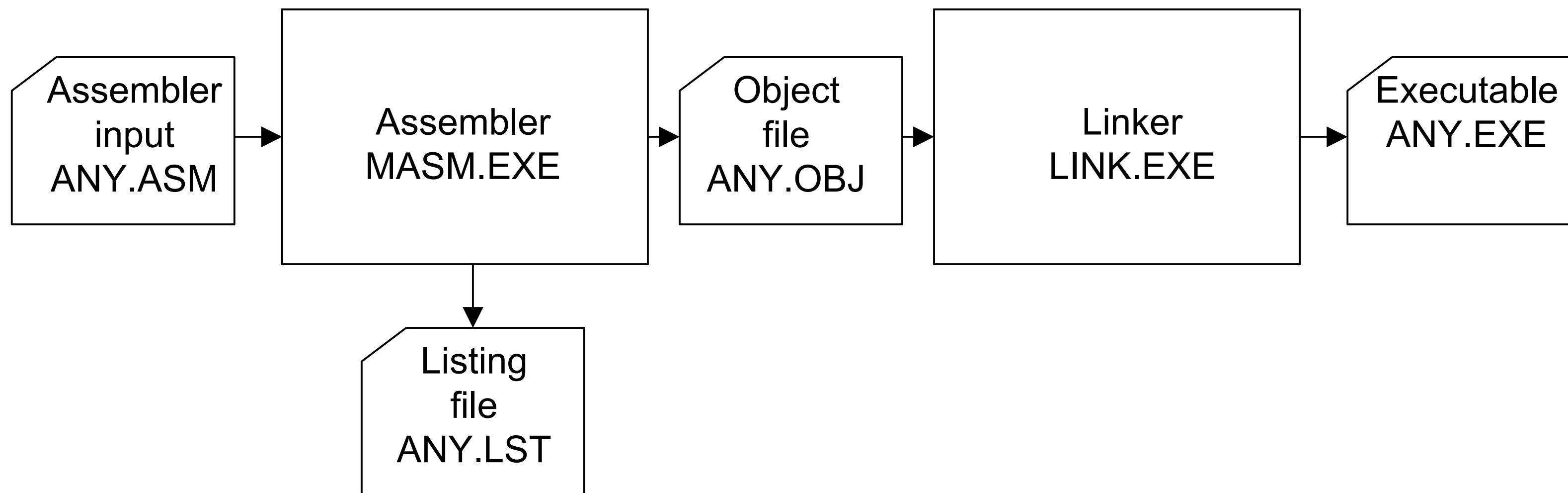
```
/* POWER2.C */
#include <stdio.h>
 int power2( int num, int power );
void main( void ) {
            printf( "3 times 2 to the power of 5 is %d\n", \ power2( 3, 5) ); }
int power2( int num, int power ) {
__asm {
            mov eax, num            ; Get first argument
            mov ecx, power          ; Get second argument
            shl eax, cl    ; EAX = EAX * ( 2 to the power of CL ) }
 /* Return with result in EAX */ }
```

A code example from another text

```
    .MODEL SMALL
             .DATA
        ORG     7000H
POINTS  DB      16 DUP(?)         ;save room for 16 data bytes
SUM     DB      ?                 ;save room for result

        .CODE
        ORG     8000H
TOTAL:  MOV     AX,7000H          ;load address of data area
        MOV     DS,AX             ;init data segment register
        MOV     AL,0              ;clear result
        MOV     BL,16             ;init loop counter
        LEA     SI,POINTS         ;init data pointer
ADDUP:  ADD     AL,[SI]           ;add data value to result
        INC     SI                ;increment data pointer
        DEC     BL                ;decrement loop counter
        JNZ     ADDUP             ;jump if counter not zero
        MOV     SUM,AL            ;save sum
        RET                       ;and return
        END     TOTAL
```

The assembly process in DOS



Comments on the assembly process

- **Our assembler**
  - **Version 6.11**
    - **Now in phase with book**
    - **Directives like .stack, .code can be used**
  - **Parts are**
    - **MASM**
      - **The assembler**
      - **A DOS program - run from the DOS prompt in start**
    - **LINK**
      - **The linker**
      - **Continue in the same DOS environment**
    - **CV**
      - **Used to do debugging work**
    - **SYMDEB**
      - **Another debugging environment**

```
stack     segment   'STACK'
          dw        100 dup (?)
stack     ends
data      segment   'DATA'
sum       dw        0
data      ends
prog      segment   'PROG"
          assume    cs:prog, ds:data
start:
          move      ax,data
          mov       ds,ax
; your code goes here
          mov       ah,4ch
          int       21h
prog      ends
          end       start

; The basic assembly program for MASM 5.10
```

The Symdeb environment

- Symdeb is a primitive debugger
  - You can use it to practice with the machine environment
  - Basic commands are
    - R - examine/modify registers
    - D - dump memory
    - U - unassemble memory
    - A - assemble
    - G - run
  - Each of these basic command groups has options and arguments
  - Your first assignment will be to build and run a program in the debugger

A code example from another text

```
         .MODEL SMALL
                     .DATA
        ORG      7000H
POINTS  DB       16 DUP(?)          ;save room for 16 data bytes
SUM     DB       ?                  ;save room for result

        .CODE
        ORG      8000H
TOTAL:  MOV      AX,7000H           ;load address of data area
        MOV      DS,AX              ;init data segment register
        MOV      AL,0               ;clear result
        MOV      BL,16              ;init loop counter
        LEA      SI,POINTS          ;init data pointer
ADDUP:  ADD      AL,[SI]            ;add data value to result
        INC      SI                 ;increment data pointer
        DEC      BL                 ;decrement loop counter
        JNZ      ADDUP              ;jump if counter not zero
        MOV      SUM,AL             ;save sum
        RET                         ;and return
        END      TOTAL
```

Using PWB – an integrated development environment

- A mystery - what is a project
  - A project is a collection of programs and libraries
  - It is the basic unit used by PWB; and also by Visual Basic and other similar products
  - PWB will take care of the details
- What is a makefile
  - Directions for making something
  - It includes dependencies so you remake only what is new or based on something new
  - Make on something up-to-date does nothing
- PWB handles most project-related items automatically
  - To start with PWB, first open a file and then run it
  - Before you even start, figure out help and window.

More PWB

- PWB and Codeview (CV.EXE)
  - PWB invokes codeview whenever you specify debug
  - Try looking at the various window options immediately
  - CV allows you to deal with
    - Registers
      - Examine
      - Change
    - Code
      - Set breakpoints
      - Step through
      - Examine
      - Run
    - Data
    - Output

What is different about the 80x86

Instructions

- MIPS is a RISC
  - 3-address code
  - All are 32 bits long
  - Load/store, branch/jump/call, arith/logical are in separate instructions
  - All operations are word-size or float/double size
  - Memory model is a simple flat space
- 80x86 is a CISC instruction set
  - Pentium is actually a RISC with a hardware 80x86 interpreter
  - 2-address code
  - Instruction length varies from 1 to 6 bytes
  - Instructions combine load/store and operations
  - Memory model is quite complicated
  - Explicit stack instructions
  - Many registers are special-purpose