Getting Started

NASM and Paul Carter's
PC Assembly Language

Get NASM

- Go to
  http://sourceforge.net/projects/nasm/files/
- Download and install appropriate binaries (Linux or Windows)
- For Mac it's a little more work:
- http://www.neuraladvance.com/2009/08/19/compiling-and-installing-nasm-on-mac-os-x/

Get Paul Carter's Source Files

- Go to
  http://www.drpaulcarter.com/pcasm/
- Download the appropriate files for your C compiler or OS
  - DJGPP
  - Linux
  - Borland
  - Microsoft
  - Cygwin (Might work for Mingw32 too)
  - Open Watcom

Build the Object Files

- After extracting sourcefiles, use make (Linux) or nmake (Windows)
- Just cd into the directory containing the files and type nmake or make at the command prompt
- Make looks for a file called Makefile and executes it when found

Look at the Makefile

- It shows what command line to use with NASM and your C compiler
- Look for ASFLAGS
- Example: Linux (gcc)

```
ASFLAGS= -f elf
  so to assemble foo.asm
  nasm –f elf foo.asm
```

- Example: Windows

```
ASFLAGS= -f win32
  so to assemble foo.asm
  nasm –f win32 foo.asm
```

Build Commands in the Makefile

- Build commands invoke a C compiler and linker to link multiple modules
- Example: Linux (gcc)

```
CFLAGS=
CC=gcc
…
prime: driver.o prime.o asm_io.o
$(CC) $(CFLAGS) -oprime driver.o prime.o asm_io.o
So to make prime use this command line
gcc -oprime driver.o prime.o asm_io.o
```

Build Commands in the Makefile

- Example: Windows (cl)

```
CFLAGS=
CC=cl
prime.exe: driver.obj prime.obj asm_io.obj
$(CC) $(CFLAGS) -Feprime driver.obj prime.obj asm_io.obj
So to make prime use this command line
cl -Feprime driver.obj prime.obj asm_io.obj
```

Driver.c

```
#include "cdecl.h"

int PRE_CDECL asm_main( void ) POST_CDECL;

int main()
{
  int ret_status;
  ret_status = asm_main();
  return ret_status;
}
```

Driver.c Compiler Differences

```
/* gcc */
int asm_main( void );
int main(){
  int ret_status;
  ret_status = asm_main();
  return ret_status;
}
/* cl */
int _asm_main( void );
int main(){
  int ret_status;
  ret_status = asm_main();
  return ret_status;
}
```

I/O Routines

- See asm code for examples. These are interfaces to the stdio C library
- **print_int**
  converts the value of the integer stored in EAX to ASCII and displays on STDOUT
- **print_char**
  print the character whose ASCII value stored in AL to STDOUT
- **print_string**
  prints out to the screen the contents of the string at the address stored in EAX. The string must be a C-type string (i.e. null or 0 terminated).
- **print_nl**
  prints a new line character to STDOUT
- **read_int**
  reads an integer from the keyboard and stores it into the EAX register.
- **read_char**
  reads a single character from the keyboard and stores its ASCII code into the EAX register.

Debugging Routines

- We're OS independent so we won't be using a GUI (or other debugger)
- We'll use the old-fashioned way: print statements in code.
- There are four debugging routines named **dump_regs, dump_mem, dump_stack** and **dump_math**; they display the values of registers, memory, stack and the math coprocessor, respectively

Dump_Regs

- **dump_regs** Prints out the values of the registers (in hexadecimal) to stdout It also displays the bits set in the FLAGS9 register. For example, if the zero flag is 1, ZF is displayed. If it is 0, it is not displayed. It takes a single integer argument that is printed out as well. This can be used to distinguish the output of different dump regs commands.

```
dump_regs 1

Register Dump # 1
EAX = 000000F6 EBX = 000000F6 ECX = 00408298 EDX = 00000003
ESI = 00000000 EDI = 00000012 EBP = 0012FF74 ESP = 0012FF54
EIP = 0040105A FLAGS = 0216             AF PF
```

Dump_mem

- This macro prints the values of a region of memory (in hex) and also as ASCII characters. It takes three comma delimited arguments. The first is an integer that is used to label the output (just as dump regs argument). The second argument is the address to display. (This can be a label.) The last argument is the number of 16-byte paragraphs to display after the address. The memory displayed will start on the first paragraph boundary before the requested address.
- **dump_mem 2, outmsg1, 1 ; dump out memory**

```
Memory Dump # 2 Address = 00408058
00408050 75 6D 62 65 72 3A 20 00 59 6F 75 20 65 6E 74 65 "umber: ?You ente"
00408060 72 65 64 20 00 20 61 6E 64 20 00 2C 20 74 68 65 "red ? and ?, the"
```

Dump_stack

- This macro prints out the values on the stack. The stack is organized as double words and this routine displays them this way. It takes three comma delimited arguments. The first is an integer label (like dump_regs). The second is the number of double words to display below the address that the EBP register holds and the third argument is the number ofdouble words to display above the address in EBP.

```
dump_stack 3,8,8
Stack Dump # 3
EBP = 0012FF74 ESP = 0012FF74
+16  0012FF84  00401586
 +12  0012FF80  0012FFC0
  +8  0012FF7C  004082A0
  +4  0012FF78  004010E9
  +0  0012FF74  0012FF80
  -4  0012FF70  00000008
  -8  0012FF6C  00000008
 -12  0012FF68  00000003
 -16  0012FF64  004010BF
```

Dump_math

- This macro prints out the values of the registers of the x87 coprocessor. It takes a single integer argument that is used to label the output

```
dump_math 1
Math Coprocessor Dump # 1 Control Word = 027F
  Status Word = 3800
ST0: 2
ST1: Empty
ST2: Empty
ST3: Empty
ST4: Empty
ST5: Empty
ST6: Empty
ST7: Empty
```