A SIMPLE PROGRAM

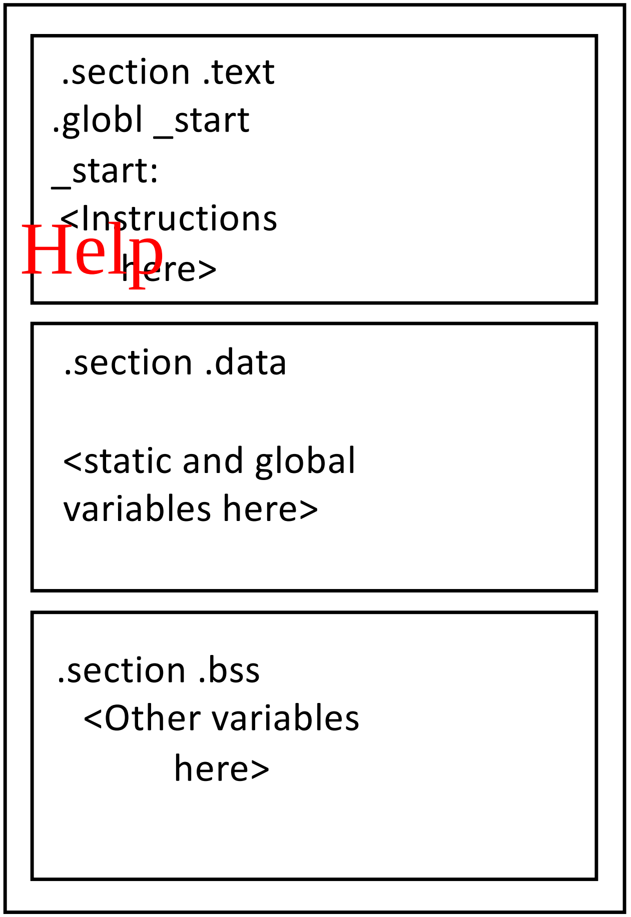
SEC204

Overview

- Sections of a program
- Cpuid instruction
- Building, running, debugging

SECTIONS OF A PROGRAM

- **.section .text**
- The text section contains instructions
- Start of the program is defined by the **_start** label.
    - This indicates the first instruction from which the program should run. If the linker cannot find it, it will produce an error
- **.section .data**
- The data section contains static and global variables (data elements with a static value, variables accessible to all program functions)
- **.section .bss**
- The bss section contains other variables
- We'll talk about the stack and heap later on



.section .text
.globl _start
_start:
<Instructions
here>

.section .data

<static and global
variables here>

.section .bss
<Other variables
here>

EXAMPLE PROGRAMME – CPUID INSTRUCTION

- Let’s create a simple program running a single instruction, cpuid
- The cpuid instruction
  - displays information about the processor
  - the EAX register is used as input to define the type of information needed
  - EBX, ECX, EDX registers display the output

EAX Value	Output
0	vendor ID string, and the maximum CPUID option value supported
1	Processor type, family, model, and stepping information
2	Processor cache configuration
3	Processor serial number
4	Cache configuration (number of threads, number of cores, and physical properties)
5	Monitor information
80000000h	Extended vendor Id string and supported levels
80000001h	extended processor type, family, model, and stepping information
80000002h-80000004h	Extended processor name string

CPUID.S

- Lets create a new file cpuid.s with the following contents (note that **$ signifies a static value** and **% signifies a register** – don't worry about understanding other instructions just yet)

```
#cpuid.s a sample program to extract
#the processor vendor Id
.section .data
output:
     .ascii "The processor Vendor ID is
'xxxxxxxxxxxxx' \n"
.section .text
.globl _start
_start:
       movl $0, %eax
       cpuid
       movl $output, %edi
       movl %ebx, 28 (%edi)
```

Cont…

```
    movl %edx, 32 (%edi)
    movl %ecx, 36 (%edi)
    movl $4, %eax
    movl $1, %ebx
    movl $output, %ecx
    movl $42, %edx
    int $0x80
    movl $1, %eax
    movl $0, %ebx
    int $0x80
```

BUILDING AND RUNNING THE PROGRAM

1. Building the executable

```
$as -o cpuid.o cpuid.s
$ld -o cpuid cpuid.o
```

2. Running the executable

```
$./cpuid
The processor Vendor ID is ‘GenuineIntel’
```

DEBUGGING WITH GDB

1. Reassemble the code using gstabs parameter (provides extra info that gdb will need)

```
$as -gstabs -o cpuid.o cpuid.s
$ld -o cpuid cpuid.o
```

2. Running gdb

```
$gdb cpuid
(gdb)run
```

3. Breaking at start, then step by step with ‘next’ or ‘step’. Once enough steps are run, execute the remaining program with ‘cont’

```
(gdb)break *_start
(gdb)run
(gdb)next
(gdb)next
(gdb)cont
```

VIEWING REGISTERS AND MEMORY

Display the value of all registers

`info registers`	Displays the values of all registers

Display value of a specific register from the program: ie %eax

`print /x $eax`
`print /d $eax`
`print /t $eax`
	Displays the value of eax in hexadecimal
	Displays the value of eax in decimal
	Displays the value of eax in binary

Display the contents of a specific memory location

`x /nyz`
 	
For example:
`x /42cb &output`
	Displays **n** number of fields,
	z size of field to be displayed (**b** for byte, **h** for 16-bit half word, **w** for 32-bit word)
	y output format (**c** for character, **d** for decimal, **x** for hexadecimal),
	
	Displays 42 bytes of the output variable in character mode
	The & indicates this is a memory location

TASKS

- After you create the cpuid file, assemble it and link it to the object file. Then run it to see the output
- Reassemble the file with gstabs, link it to the object file. Run the program in debug mode.
- Create a breakpoint at start, then run it step by step
- Display the value of registers %eax register before cpuid instruction executes
- Display the value of registers %ebx, %edx, %ecx after cpuid executes.
- Display the values of registers %ecx, %edx in ascii after the output string is displayed

Using printf

- Lets modify the cpuid.s file to include the C function printf

```
#cpuid2.s View the CPUID Vendor ID
#string using C library calls
.section .data
output:
     .asciz “The processor Vendor ID is
‘%s’ \n”
.section .bss
     .lcomm buffer, 12
.section .text
.globl _start
_start:
      movl $0, %eax
      cpuid
```

Cont..

```
   movl $buffer, %edi
   movl %ebx, (%edi)
   movl %edx, 4 (%edi)
   movl %ecx, 8 (%edi)
    pushl $buffer
    Pushl $output
    call printf
    addl $8, %esp
    Pushl $0
    Call exit
```

BUILDING AND RUNNING THE PROGRAM

1. Building the executable

```
$as -o cpuid.o cpuid.s
$ld  -dynamic-linker /lib/ld-linux.so.2 -o cpuid -lc cpuid.o
```

2. Running the executable

```
$./cpuid
The processor Vendor ID is ‘GenuineIntel’
```

DEBUGGING WITH GDB

1. Reassemble the code using gstabs parameter (provides extra info that gdb will need)

```
$as -gstabs -o cpuid.o cpuid.s
$ld  -dynamic-linker /lib/ld-linux.so.2 -o cpuid -lc cpuid.o
```

2. Running gdb

```
$gdb cpuid
(gdb)run
```

3. Breaking at start, then step by step with ‘next’ or ‘step’. Once enough steps are run, execute the remaining program with ‘cont’

```
(gdb)break *_start
(gdb)run
(gdb)next
(gdb)next
(gdb)cont
```

FURTHER READING

- Professional Assembly Language, chapters 3, and 4
- Reference information on IA 32: http://www.sandpile.org/