Assembly Programming HOWTO

Download all the required stuff

- 1. Download VirtualBox for <u>Windows or Mac</u>.
- 2. Download the VirtualBox extension pack.
- 3. Download the <u>VM for Assembly Programming</u>.

Install VirtualBox and the VM

- 1. To install VirtualBox, just doubleclick on the EXE and follow instructions on the screen. On Mac, double click the DMG file, then drag VirtualBox into Applications.
- 2. Run VirtualBox from your Start/Applications Menu.
- 3. Go to File > Settings > Extensions. Click on the Add Package button on the right. Select the extension pack you downloaded earlier. Follow onscreen instructions to complete installation.
- 4. Go to File > Import Appliance.
- 5. Point to the .OVA file you downloaded above. This will usually be in your Downloads folder.
- 6. Once the Import is complete, select AssemblyProgramming from the list of VMs and click on Start.
- 7. Your VM should now boot. If you get some popups in blue on the top, you can safely close them.

Begin Assembly Programming with Eclipse

Eclipse should autostart as your VM boots up. Two sample projects exist, one with a C program and one with an ASM program. Open the Test project:

C/C++ -	Eclipse Pla	tform							\bowtie	1↓ 	12:00 AM 👤	Student	÷
	File Edit	Source Refactor Navigate S	earch Run	Project	Window	w Help							
0] 📬 🛛 🖓	🖹 📄 🔂 v 🗳 v 🕞 v	% × ⊗ ×	ॐ▼ 0▼	Q	۵ 🖋 🖻] § ▼ ∛ ▼ *>	, , , , , , , , , , , , , , , , , , ,			🖹 🖪 🕻	/C++	**
>_	Project	Explorer 🛛 📄 🔩 🔻 🖻	· 🗆						- 0	E Outlir	n 🕱 💿 Ma	ke 🗧	
ل	👕 Test_									Anoutlin	e is not availal	ole.	
	👕 Test	New	+										
		Сору	Ctrl+C										
		Paste	Ctrl+V										
		<u>D</u> elete	Delete										
		Move											
		Rename	F2										
1		Import											
		Export											
		Op <u>e</u> n Project											
		Show in Remote Systems view											
		Run As	•										
		<u>D</u> ebug As	Þ										
100		<u>P</u> rofile As	•										
		T <u>e</u> am	•										
		Comp <u>a</u> re With	•	()	(
-		Restore from Local History		olems 🖉 T	Tasks	🚽 Console 🛱 🖉 🔲	Properties				1 E E	7 🖸 ₹	- 8
_		Properties	Alt+Enter	soles to dis	splay at	t this time.							
0													
	•	🖆 Test											

After opening the project, open the associated program file, in this case *test.c.* You can now toggle breakpoints at specific lines of code. Breakpoints pause the program execution so that we may view values of program variables:



Now we have to Build our code, which is just a fancy name for compiling it. Select Project > Build All.

C/C++ - T	est/test.c - Eclipse Platform				× •	🏚 🜒) 12:01 AM 👤 Student 🖏
	File Edit Source Refactor Navigate Sear	h Run	Project Window Help			
•	📑 - 🔚 🐚 🎃 🖬 💣 - 🎯 - 🞯 -	« • (Open Project			😭 📴 C/C++ 🔉
			Close Project			
>_	Project Explorer 🛛 🕒 🤹 🖓	.c test	Build All Ctrl+B			
	🔻 😂 Test	int	Build Configurations		Â	↓ª _z 😿 🖋 🖷 🗮 🏹
	▶ 🔊 Includes		Build Project			main():int
	Debug		Build Working Set			
No contraction	P C Lest.c		Clean			
	lestAsm	1	✓ Build Automatically			
		1	Make Target 🔸			
			Properties			
				1		
_						
		(1)		0		
		🔡 Prol	blems ⁄ Tasks 💷 Console 🕱 🖉 🖿 P	roperties		e 📃 🗸 📑 🖬 🗖
		No con	soles to display at this time.			
			Writable	Smart Insert 1 · 1		
			vviitable	Sindremsere 1.1		

In this case, you are using a sample program provided by me in working condition. Therefore the Build will go through painlessly. Once you write your own programs, you will see a number of errors showing up here. These errors will help you fix your compilation errors.

Just because a program compiles does not mean it is error free. Now we will debug the program to see if it actually does what we intended it to do.



To start debugging, go to Run > Debug in the menu.

Eclipse will now change the *perspective* to *debug mode*. You will see a number of windows with all kinds of information about your program.

Debug -	Test/test.c - Eclipse Platform		🖂 💌 tļ	🜒) 12:25 AM 👤 Student 🔱
	File Edit Source Refactor Navigate Search Run Project Window Help			
0] 🗂 र 📓 📾 📾 📓 🕸 र 📭 💁 🎥 🥔 🖉 र 🕽 🖉 🖉 🖉 र 🖓 र	~		🖹 🍄 Debug 🏻 🎽
>_	🇚 Debug 🛿 🦂 🦓 🕼 🕪 💷 👪 🐉 🗫 🖘 📖 🖬 😥 🍸 🗖 🗖	🕬= Variables 🤷 Breakpoints	🕷 Registers 🛛 🛋 Modu	iles 🗖 🗖
	▼			🖄 🐗 🖻 📑 🖻 🎽
	[®] /home/student/workspace/Test/Debug/Test [2161] [cores: 0] [®] Thread [1] 2161 [core: 0] (Suspended : Breakpoint)	Name	Value	Description
	≡ main() at test.c:2 0x80483ba	iiii ebp	0xbffff3d8	Ų
	jgdb	QC))))
		Name : ebp		
	€ test.c ¤		🗖 🗖 🔡 Outline 🚟 Di	isassembly 🛿 🗖 🗖
	<pre>int main() {</pre>		<u>A</u>	~
	<pre>int a = 2; int b = a * 3;</pre>		Enter locatio	n here 🚽 🔊 💩 🔄 🔯
	<pre>int c = &a</pre>		♦ 080483ba:	movl \$0x2 -0xc(%ebp)
	b += 5;		[∞] 3	int b = a * 3;
	return a;		080483c1: 080483c4:	mov -0xc(%ebp),%edx mov %edx.%eax
124	1		080483c6:	add %eax,%eax
14			08048308:	add %edx,%eax
	🗟 Console 🕱 🖉 Tasks 🕵 Problems 🕥 Executables 🔋 Memory			:
	Test [C/C++ Application] Test			
9)))
	□ ◆			

Here is a quick review of all the things you see on this screen:

- 1. The code file, **test.c**
- 2. **Disassembly** of the C program with the corresponding assembly statements.
- 3. Registers
- 4. Variables
- 5. **Memory**. More information on accessing the right memory locations is given below.

It is interesting to observe how a program modifies memory. However, to know where to look in all the memory the computer has, you must first know where your program is storing useful information. This is easy to find using disassembly and registers.

First, note that the first line of C code is (highlighted in light green colour on the left)

int a = 2;

Debug -	Test/test.c - Eclipse Platform		🖂 💌 🕇 🖛)) 12:35 AM 👤 Student 🔅
	File Edit <u>S</u> ource Refac <u>t</u> or Navigate Search Run Project Window Help			
0	🕄 ▼ 🔛 🖄 🚔 📾 📾 🗱 🕸 - Q ▼ Q ▼ 🛛 🥭 🥭 🔗 ▼ 🛛 🥖 🖓 ▼ 🖓 → 🖓 ▼ 🐎 💠	• => =		腔 梦Debug »
>_	🎋 Debug 🛿 🦷 🕷 🕪 💷 🛤 🔹 🕫 🖄 🖬 🕅	🗖 🗖 🕬= Variables 🤷 Breakpoir	nts 🛲 Registers 🕱 🛋 Module:	s
	▼ E Test [C/C++ Application]			約 🐗 🖻 📑 👻 🎽
	/home/student/workspace/Test/Debug/Test [2216] [cores: 0]	Name	Value	Description
	main() at test c:2 0x80483ba	and the second s	0xbffff3d8	Ů
	adb	1010 •		
	y goo	Name : ebp		
		(()))
1	Itest.c ☎libc_start_main() at 0x14c4d3		🗆 🗖 🚟 Outline 🚟 Disa	assembly 🛛 🗖 🗖
	int main() {		Â	∇
	a = 2; int b = a * 3;		Enter location h	here 🔻 🔊 🏠 🚖 💽
	$int c = \delta_a;$			ovl \$0x20xc(%ebp)
	b += 5;		<i>⊷</i> 3	int b = a * 3;
	return a;		080483c1: mo 080483c4: mo	ov -0xc(%ebp),%edx
100	I		080483c6: ad	id %eax,%eax
14			080483c8: ad	1d %edx,%eax
	Console 🖉 Tasks 🕅 Problems 💟 Executables U Memory 🛛			
_	Monitors 📲 💥 🙀 0xbffff3d8:0xBFFFF3D8 <hex> 🖾</hex>	🕈 New Renderings		10
	◆ 0xbffff3d8 Address 0 - 3 4 - 7	8 - B C - F		
	BFFFF3D0 E0830408 00000000	0 00000000 D3C41400		U
	BFFFF3E0 01000000 74F4FFB	r /tr4rrbr 00F9FFB/		J

See the disassembly window. Observe the corresponding assembly line (highlighted in light green colour on the right):

movl \$0x2, -0xc(%ebp)

The above line indicates that the program has been compiled such that the variable "a" is stored in the stack at (EBP-0x0c).

Debug -	Test/test.c - Eclipse Platform			🖂 💌 tļ	🜒) 12:40 AM 👤 Student 🔱
	File Edit <u>S</u> ource Refac <u>t</u> or Navigate Search	Run Project Window Help			
	☐ 🕄 ▼ 🗟 👜 📾 📄 🕸 ▼ O ▼ 🂁 🖉	♥∥♥]♪]♪ ♥ ♥ ♥ ♥ ♥	▼		腔 梦Debug *
	🗱 Debug 🛛 🔌 🦓 🕪 💷 .	N 🔍 👁 🖄 👼 🖬 🕱 🎽 🗖	🕬= Variables 🤷 Breakpoints	🖁 🛲 Registers 🛿 🛋 Modu	iles 🗖 🗖
	▼ © Test [C/C++ Application]	(freed) (約 📲 🖻 🖻 🎽
	/home/student/workspace/Test/Debug/Test / home/student/workspace/Test/Debug/Test / home/student/workspace/Test/Debug/Test / home/student/workspace/Test/Debug/Test/ / home/student/workspace/Test/ / home/student/workspace/Test/ / home/student/workspace/Test/ / home/student/workspace/Test/ / home/student// / home/stu	it [2332] [cores: 0] akpoint)	Name	Value	Description
	≡ main() at test.c:3 0x80483c1		iiii ebp	0xbffff408	
	📓 gdb		(())))
			(4())
	libc_start_main() at 0x14c4d3	•		🗖 🗖 🔡 Outline 📟 Di	isassembly 🛛 🗖 🗖
	<pre>int main() {</pre>			8	~
	int a = 2; int b = a * 3;			Enter locatio	n here 🖃 🔊 🗞 🕞 🔯
	$int c = \delta_a;$				
	b += 5;			- 3	int b = a * 3;
	return a;			⇒ 080483c1:	mov -0xc(%ebp),%edx
100	1			080483c6:	add %eax,%eax
				080483c8:	add %edx,%eax
)))
~ <u></u>	📮 Console 🤄 Tasks 🖹 Problems 🚺 Executable	es 🚺 Memory 🕱		1012 1012	
	Monitors 🕂 🙀 🙀	0xbffff3fc : 0xBFFFF3FC <hex> 🛿 🕂</hex>	New Renderings		
	♦ 0xbffff3fc	Address 0 - 3 4 - 7 8 -	- B C - F		()
		BFFF5F0 80F21100 00000000 ES	9830408 02000000		Ĵ
0		BFFFF400 E0830408 00000000 00	0000000 D3C41400		
	□◆				

Now, in the Registers tab (see image above), note that the value of EBP is 0x3ffff408. So we have to open the location 0x3ffff408 - 0x0c = 0x3ffff3cc in memory. In order to do that, first click on the <u>Memory</u> tab, then click on the green plus (+) icon. Now enter the memory address in the dialog box that appears as shown below.

Eclipse						🖂 💌 👣	, 🜒) 12:33 AM 👤 Stu	dent 🔱
	File Edit Source Refactor Navigate Search Run Project V	Vindow Help						
•	」 C1 ▼	• {} • \$ \$ \$ \$	·				🗈 🕸 Debu	» pu
	🕸 Debug 🛿 🦂 🍇 🗱 🕪 💷 📕 🧏 👁 🔅 🗏	₅ i+ ≳ ▼ " □	🕬= Variables 🤷 B	Breakpoints 🖁	Register	s 🛱 ڬ 🛤 Mo	dules	- 0
	▼						🐌 🏘 🖻 📑	
	/home/student/workspace/Test/Debug/Test [2161] [cores: 0]		Name	١	Value		Description	Ê
	* B* Thread [1] 2161 [core: 0] (Suspended : Breakpoint)		1010 ebp		oxbffff3dg			U
	adb		(4(0.01111300			
		😣 Monitor Memo	mory)))
	د test.c ک	Enter address or exp	pression to monito	or:	- 0	🗄 Outline 🎫	Disassembly 🛿	
	<pre>int main() { int = 2; }</pre>			~	A			▽
	int a = 2; int b = a * 3;					Enter locat	ion here 👻 👔 🦍	50
	$int c = \delta a;$	~				080483ba:	movl \$0x2,-0xc(%ebp	p) (q
	b += 5;	Cancel	ОК			∞3 080483c1+	int b = a * 3;	
	return a; }					080483c4:	mov %edx,%eax	
194						080483c6: 080483c8:	add %eax,%eax add %edx,%eax	ų.
	(4((()	
Fa	📮 Console 🖉 Tasks 🗟 Problems 📀 Executables 🟮 Memory 🕱					🗋 🛃 🖷	1010	~ - 8
	Monitors 🕂 💥 🔆							
						1		

You will now see the contents of different memory addresses starting at 0x3ffff3c0:

Debug -	Test/test.c - Eclipse Platform			🖂 🕟 👣	🜒) 12:41 AM 👤 Student 🔱
	File Edit <u>S</u> ource Refac <u>t</u> or Navigate Search	Run Project Window Help			
0	🗅 • 🔒 🚔 🚔 🕸 • O • 🏊 🤌 🥙	⋪∙∫⊿∫⋬∊∛∊ゃ ⇔	~	🖹 🏇 Debug 🏻 🎽	
>_	🕸 Debug 🛛 🦂 🎉 🕪 💷 🖉	1 🥆 🕫 🖄 🤜 🖬 🗷 🗖 🗖	🕪 Variables 💊 Breakpoints	s 🔠 Registers 🛿 🛋 Modu	les 🗖 🗖
	▼ Test [C/C++ Application]				約 🐗 🖻 📑 🖻 🎽
	/home/student/workspace/Test/Debug/Test	[2332] [cores: 0]	Name	Value	Description
	Thread [1] 2332 [core: 0] (Suspended : Step)		otot esp	UXDITIT318	
Nor	≡ main() at test.c:4 0x80483cd		iiii ebp	0xbffff408	
	pdb 📓		(4(
			(4)		
	libc_start_main() at 0x14c4d3			🗖 🗖 🔡 Outline 🕮 Di	sassembly 🛛 🗖 🗖
	<pre>int main() {</pre>				~
_	int a = 2;				
	int $c = a + 5;$			Enter location	n here 🔻 🖏 🟠 😫
				080483ca:	mov %eax,-0x8(%ebp)
	b += 5;			- 4	int c = &a
	return a;			080483d0:	mov %eax0x4(%ebp)
500	,			6	b += 5;
				080483d3:	addl \$0x5,-0x8(%ebp) 🗦
	((
·	📮 Console 🙆 Tasks 🔝 Problems 🚺 Executables	🗴 🕼 Memory 🕱		1000	💀 🎚 🔄 🖏 ⊽ ⊽ 🗖 🗖
	Monitors 🕂 🙀 🧕	xbffff3fc : 0xBFFFF3FC <hex> 😫 🔮</hex>	• New Renderings		
	♦ 0xbffff3fc	ddress 0 - 3 4 - 7 8	- B C - F		
		BFFF5F6 80F21100 00000000 E	9830408 02000000		
0		BFFFF400 A 06000000 00000000 00	0000000 D3C41400		
	·				

Observe that the contents of the memory address 0x3fff3cc is 0x0000002. This corresponds to the first statement in the C program: int a = 2;

Memory addresses are a *word* long. In this particular example (32 bit x86 processor running Linux), each word is four bytes long. Therefore the variable a is stored in addresses 0x3ffff3cc to 0x3ffff3cf. Registers such as EBP and EDX are also a word long.

In the image above, you can also see that address 0x3ffff400 is highlighted in red. why is that so? The next line of this code is as follows

int b = a * 3;

In the Disassembler window, you can see that the corresponding assembly code is:

```
080483c1: mov -0xc(%ebp),%edx
080483c4: mov %edx,%eax
080483c6: add %eax,%eax
080483c8: add %edx,%eax
080483c8: mov %eax,-0x8(%ebp)
```

(The values 080483c1 etc. here correspond to the location in memory where the particular assembly instruction exists.)

As you can see, first we copy (mov) the contents of memory address EBP-0x0c to register EDX, so EDX has the value 2. We then copy the contents of EDX to EAX. We then add EAX to EAX, then EDX to EAX. Finally, we copy EAX to EBP-0x08. EBP-0x08 is where the variable b is stored.

We asked to multiply a by 3 and store in b. Instead, it added a thrice and stored it in b.

This is known as *compiler optimization*. The compiler (in this case gcc, the GNU C Compiler), decided that it would be more efficient for this processor to add it thrice, rather than multiply it by 3 once.

Now, let's revisit the red text. What the debugger is doing is to highlight for us the memory contents that got changed by the instruction that was just executed. But don't programs run all at once? Not in a debugger! Remember we added a breakpoint when we first saw the C program. The program had stopped at that location (after int a = 2;). Before I took the screenshot, I selected Run > Step Into from the menu first. This caused the debugger to run the next statement (int b = a * 3;), highlight in red what changed, and stop again. Now we can press F5 (the shortcut key for Step Into) and get the next statement to run. On each statement run, certain registers or memory contents should change.

Eclipse supports various "*views*". To go back to code view. Select Run > Terminate from the menu. Then, click on the >> button on the top right, then click on C/C++.

	🛛 🖾 ti	♠)) 8:22 AM	4 👤 Student	₩
		Ē	🅸 Debug	1 "
		ž ≉t,⊟	C/C++	
[Description		Resource	A
)	
				•
Enter location he	re 🔻 🔊	à (5)	1	
				A
				8

Working with an Actual Assembly Program

So far, we've looked at the disassembly of a C program. Now let's write an actual assembly program.

First, right click on Test in the Project Explorer and select Close Project. Now, right click on TestAsm and select Open Project. Double click on the file testasm.S.

From here on, everything is pretty much the same as before. You will see some assembly code. Go ahead and add a breakpoint. Now Build All, and Run. You should see something like this:

800	C/C++ - TestAsm/testasm.S - Eclipse Platform File Edit Source Refactor Navigate Searc	i h Run Project Window Help	× 100) 1 . ()) 8:4	SAM 👤 Student	ψ
۲	1 1 - E C 🚔 🗎 付 - C - C -	δ τ 8τ βτ Οτ δ τ β τ α τ α τ β τ φ τ φ τ φτ			🖬 🖾c/c++	»
	🕒 Project Explorer 😫 🛛 🖻 🍓 🍷 🗖 🗖	E testasm.5 B		E Outlin 23	🖲 Make 🗧	• 🗆
	in Text V ≦ TextAum + § Binaries + § Dincludes + § Dincludes + § texterm S	<pre>section .dets message:.string "Hello Worldl\n" .section .text # this directive allows the linker to see the "main" label # which is our entry point .globl main # this directive allows the eclipse gdb to see a function called "main" .fmess for \$4, Yeax mov \$5, Yeax</pre>		∞ messag ∞ main	A	4
		IL Problems @ Tasks @ Console X \ D Properties	36	* 🔐 💭 🖉	1 Q + 13 + 5	. 9
		<pre>-terminated>TestAsm [c/C++ Application]/home/student/workspace/TestAsm/Debug/TestAsm (1/27/14 B:45 AM) Hello World!</pre>				
0						
	0*	Writable Smart Insert 2:27				

Down below in the console, you can see the output: Hello World. Now let's debug this program (although there aren't any real bugs). Go to Run > Debug. Your screen should change to something like this:

800	Debug - TestAsm/testasm.S - Eclipse Platform			🛛 🖾 🛤 🛊 🐠) 8:51 AM 👤 Student 🛟
۲	File Edit Navigate search Run Project Window Help			🖬 🏷 Debug 👋
	Pebug 23 後 10 回 10 元 三 10 元 10 元	🕬= Variables 💁 Breakpoints 1 Registers 😫	Modules	8 🐗 🖨 📑 🐨 🏱 🖻
<u> </u>	TestAsm [C/C++ Application]	Name	Value	Description
	🔻 🎲 /home/student/workspace/TestAsm/Debug/TestAsm [1968] [cores: 0]	🕮 ebx	3063796	
	Thread [1] 1968 [core: 0] (Suspended : Breakpoint) main() at testacm 5:13 0x80403b4	ዛዝ esp	0xbffff3bc	
X	J gdb			
	is testasm.S 23		📟 🗖 🔡 Outline 📟 Disassembl	v #
	section .data		Enter loc	ation here 🚽 👔 🏠 🚱 💽 📑 💌 🏹
	message: .string "Hello World!\n"		080483a6; je 0x804	83b1 <frame dummy+33=""/>
	.section .text		080483a8: movl \$0x	8049f24,(%esp)
	# this directive allows the linker to see the "main" label		080483b1: leave	**
	.globl main		080483b2: ret 080483b3: nop	
1	# this directive allows the eclipse gdb to see a function called "main"		,∞13 mov \$4, %e main:	ax
	.func main main:		♦ 080483b4: mov \$0x4	, Seax
Q.	9 mov \$4, %eax		080483b9: mov \$0x1	,%ebx
	mov \$message, \$ecx		15 mov \$messa 080483be: mov \$0x8	ge, %ecx 04a010,%ecx
	mov \$14, %edx int \$0x80		16 mov \$14, % 080483c3: mov \$0xe	edx Sedx
	mov \$0, %eax		17 int \$0x80	0
			18 mov \$0, %e	ax
			080483ca: mov \$0x0 080483cf: nop	,%eax
			eseasadelibc_csu	_init:
				(4(
	🖾 Console 🖉 Tasks 🖹 Problems 💽 Executables 🚺 Memory 🛙			📑 🛃 💷 🐖 🚳 🏭 😵 🖷 🗖
	Monitors 🔶 🙀 🙀 Ox804a010: 0x804A010 <hex> 🕄 🕂 New Renderings</hex>			
	+ 0x804a010 Address 0 - 3 4 - 7 8 - B C - F			
	68844618 48656C6C 6F28576F 726C6421 648868688 68844628 60886688 60886688 60886688 60886688 60886688			
	0804A030 00000000 00000000 00000000 00000000			
	6894A649 600066000 600066000 600066000 600066000			
	08044050 00000000 00000000 00000000 00000000			
0	88044608 6006609 6006609 6006609 6006609 6006609 6006609 6006609			T
-	۵,			

The disassembly on the right is pretty clean. Let's zoom in on that.



Observe that \$message has been replaced with 0x804a010. The variable message is actually nothing but an address in memory where the string "Hello World!\n" is stored. In C programming we call this a pointer. It is not the value of the pointer that is important... it is what the pointer points to that is of interest to us. So let's go look at this memory address:

es 🚺 Memory 🛛								
🛖 🙀 🙀 0x804a010 : 0x804A010 <hex> 🖾 🕂 New Rende</hex>					nderings			
	Address	0 - 3	4 - 7	8 - B	C - F			
	0804A010	48656C6C	6F20576F	726C6421	0A000000			
	0804A020	00000000	00000000	00000000	00000000			
	0804A030	00000000	00000000	00000000	00000000			
	0804A040	00000000	00000000	00000000	00000000			
	0804A050	00000000	00000000	00000000	00000000			
	0804A060	00000000	00000000	00000000	00000000			
	<u>08044070</u>	0000000	0000000	0000000	0000000			

Starting with $0 \times 0804A010$, you can see the string "Hello World!\n" in all its hexadecimal glory. Let's pull up an <u>ASCII chart</u> here to decode what this is. I'll do the easy ones. $0 \times 0804A015$ is 0×20 , which is a space. $0 \times 0804A01B$ is 0×21 , which is an exclamation mark. $0 \times 0804A01C$ is $0 \times 0A$, which is a new line character.

Our assembly code seems to be more intuitive or clean as compared to the assembly code generated by the C compiler. This is one of the reasons sometimes people choose assembly over high level languages. Observe how the compiler converted a multiplication to three sets of additions. The compiler is a computer program too, and occasionally does not necessarily generate the cleanest or most efficient assembly code (we have discussed in class). Where execution efficiency is of importance, assembly programming is your best bet. Some examples are Operating System code, code that interacts with external hardware in a time sensitive manner (think pacemaker or other medical devices), or scientific formulae that must run zillions of times on a supercomputer (matrix multiplication). A huge amount of mathematical formulae are handcoded in assembly by expert programs, and licensed at very high cost to run on supercomputers that predict the weather or help design new drugs.

Writing your own Assembly Program

Now that you are comfortably running the sample programs I've given you, let's get you started on your own first project. First, close any projects you've got open.

Go to File > New > C++ Project

Select Empty Project, Linux GCC, and give your project a name. Click on Finish.

Go to Project > Properties > C/C++ Build > Settings from the menu. Click on GCC Assembler. Under Commandline Patterns, you will see:

```
${COMMAND} ${FLAGS} ${OUTPUT_FLAG} ${OUTPUT_PREFIX} ${OUTPUT}
${INPUTS}
```

Change this to:

```
${COMMAND} ${FLAGS} -g --gstabs ${OUTPUT_FLAG} ${OUTPUT_PREFIX}
${OUTPUT} ${INPUTS}
```

This change asks the assembler to include debugging symbols in the compiler binary. This allows the debugger to step through code.

Now create your first program. Do File > New > Source File. Give your program a name, remember to end it with .S. That's a capital S, do not put in a .s or .asm.

Now type in your first program, save, build all, fix your compilation errors, and when the build succeeds, run and debug.

What to do when things don't work?

Sometimes, Eclipse will get stuck. You might end up closing so windows or move them around, and then you won't be able to see your registers any more. Or worse...

Here is how to avoid such problems:

- 1. Try not to play with Eclipse, yet. It is a finicky bit of software.
- 2. Remember that you can import the .ova file I've given you, as many times as you wish. So if things break, just start afresh.
- 3. If you start afresh, you'll loose all the code you've written in the VM. Your options are:
 - a. Email the code from the /home/student/workspace directory to yourself using the Firefox browser.
 - b. VirtualBox allows you to share a directory between your host Operating System (Windows/Mac) and Linux. Copy all your code to your OS that way.
- 4. Delete the /home/student/.eclipse and /home/student/eclipse folders in Linux. This will make Eclipse forget most things and start afresh.

If you'd like to email me, it is best done with a screenshot of your VM. Go to Machine menu in VirtualBox, and select Take Screenshort. Email the generated PNG file to me.

How to Share files between the VM and your laptop's host OS

The Internet works within your VM, so you can use the browser to open your email and send and receive files. In case you want to share files between your VM and your laptop's host OS (Windows/Mac), here's how to do that.

First, in your VM, open the terminal. Type the following:

student@ubuntu:~\$ sudo nano /etc/group

The password is **student**.

At the end of this file, type the following:

vboxsf:x:999:student

How do you go to the end? Just use the down arrow keys. How do you save and exit? Press Control-o, then Control-x.

Now Shutdown the VM. Click on the gear icon on the top right, and select Shut Down..

Next, in VirtualBox, go to the Settings for this VM. The last option is Shared Folders. In that click on the little folder-plus icon to the right. Select a Folder Path of your preference, and remember to check Auto Mount before you hit OK.



That's it, you're done. Now Start the VM again. Click on the folder icon on the left, then File System > media. You should see your shared folder there. Make sure you are able to copy files both from and to this folder.

Home Folder					🛛 💌 🗊 👣 🜒 11:0:	। AM 👤 Student 🔱
GNU nano 2.2.6	808 /	_				
<pre>hetdev:x:113: whoopsie:x:114: mlocate:x:115:</pre>	Eomputer Home Desktop	media sf_neha	Documents		← → Q Search	
ssh:x:116: avahi-autoipd:x:117: avahi:x:118:	Documents	bin	boot	dev	etc	
pulse:x:119: pulse-access:x:120: utempter:x:121: rtkit:x:122:	Music Pictures I Videos	home	lib	lost+found	media	
saned:x:123: student:x:1000: sambashare:x:124:student vboxsf:x:999:student	E File System	mnt	opt	proc	root	
	Browse Net	run	sbin	selinux	SIV	
AG Get Help AD WriteOut AR AX Exit AJ Justify AM		sys	tmp	usr	var	
		initrd.img	1 101 1010 vmlinuz			
•				"media" sele	cted (containing 3 items)	