

mp2 Warmup Instructions (Updated 3/2/2020 by Ron Cheung for building using the Linux environment)

Study the lecture notes on the tools and instruction set. Then follow along with this document. Make sure everything works for you as it is shown here and that you understand **everything**. Turn in your work on this "warmup" along with the rest of your mp2 assignment.

Here's your first snippet of assembler. It is written in Intel assembly language as follows:

```
movl $8, %eax
addl $3, %eax
movl %eax, 0x200
```

Let's see how to get this to run on the tutor VM. Since it only uses registers and a memory location, it doesn't need any "startup" module. We just have to get these instructions into memory and execute them.

1. You can find tiny.s in mp2/warmup/. Copy the warmup directory to your mp2 directory which has the file tiny.s:

```
.globl _start
_start:
    movl $8, %eax
    addl $3, %eax
    movl %eax, 0x200
    int $3
.end
```

I've added the "int \$3" to trap back to Tutor at the end. Note also that I have used the pseudo-op .end to tell the assembler that this is the end of the code to be assembled.

2. Build a 32-bit executable by running the assembler as --32 and then the loader ld -m elf_i386. Normally we would put these commands in a makefile, but here you want to become familiar with the individual steps.

```
itserver6$ as --32 -al -o tiny.o tiny.s
```

```
1          # tiny.s: mp2warmup program
2
3          .globl _start
4      start:
5 0000 B8080000    movl $8, %eax
6          00
7 0005 83C003     addl $0x3, %eax
7 0008 A3000200   movl %eax, 0x200
7          00
8 000d CC        int $3
9          .end
```

```
itserver6$ ld -m elf_i386 -N -Ttext 0x100100 -o tiny.lnx tiny.o
-----
```

Here the -N flag tells ld to make a self-sufficient, simple executable, and the "-Ttext 0x100100" tells it to start the code area at 0x100100

3. We can look at the contents of tiny.lnx with the help of objdump. To get the hex contents as well as the disassembly, use "-S" option:

```
-----
itserver6$ objdump -S tiny.lnx

tiny.lnx:      file format elf32-i386
```

Disassembly of section .text:

```
00100100 <_start>:
 100100:      b8 08 00 00 00      mov     $0x8,%eax
 100105:      83 c0 03           add     $0x3,%eax
 100108:      a3 00 02 00 00      mov     %eax,0x200
 10010d:      cc                int3

      b808000000      mov $0x8, %eax      at location 0x100100
      83c003          add $0x3,%eax      at location 0x100105,mov is 5 bytes long
      a300020000      mov %eax, 0x200    at location 0x100108,add is 3 bytes long
      cc            int $3      at location 0x10010d,mov is 5 bytes long
      .end              at location 0x10010e,int is 1 byte long
```

Later, we will cover how to encode instructions in bits, but for now it is interesting to find the 0x200 address hidden in the movl %eax, 0x200 instruction, and the 08 and 03 in the first two. Surprisingly, the 08 takes up 4 bytes but the 03 only one. The instruction set is optimized to be able to add small numbers into registers very quickly. The instruction size is important to speed because each instruction must be read out of memory before it can be executed.

4. We download and run tiny.lnx on the tutor VM, executing one instruction at a time to see how the registers change. To execute one instruction at a time, use the "t" command in Tutor, for "trace". To get started, set the EIP to 100100, pointing the CPU to address 100100 as the next instruction to execute.

```
-----
Logon to tutor-vserver VM using credentials provided. Transfer the tiny.lnx file
from ulab to the VM using scp:
```

```
tutor-vserver$ scp username@users.cs.umb.edu:cs341/mp2/mp2warmup/tiny.* .
tutor-vserver$ ls
```

you should see all the tiny.* files. Download the tiny.lnx file from tutor-vserver VM to the tutor VM using mtip:

```
tutor-vserver$ mtip -f tiny.lnx
  For command help, type ~?
  For help on args, rerun without args
```

Code starts at 0x100100
Using board # 1
(restart tutor VM and hit <CR> at vserver VM)

```
Tutor> ~downloading tiny.lnx          //enter ~d
.Done.
Download done, setting eip to 100100

Tutor> md 100100                      //Look at the code: same as above
00100100    b8 08 00 00 00 83 c0 03 a3 00 02 00 00 cc 90 90 .....
Tutor> rd
EAX=0000000b EBX=00009e00   EBP=000578ac
EDX=00101b88 ECX=00101bac   ESP=003ffff0
ESI=00090800 EDI=00101d5c   EIP=0010010d
EFLAGS=0302 (IF=1 SF=0 ZF=0 CF=0 OF=0)
Tutor> md 200                        //Check target area using md or mdd
00000200    0b 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
Tutor> ms 200 00000000                //Clear target area(8 0's for 32-bit write)
Tutor> md 200                        //Check again--OK
00000200    00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....

Tutor> rs eip 100100                 //Set initial EIP to start addr
Tutor> t                             //Trace: execute 1 instruction
Exception 1 at EIP=00100105: Debugger interrupt
Tutor> rd                           //See EIP at 100105 (i.e. offset 5), and
EAX=00000008 EBX=00009e00   EBP=000578ac    //8 now in EAX
EDX=00101b88 ECX=00101bac   ESP=003ffff0
ESI=00090800 EDI=00101d5c   EIP=00100105
EFLAGS=0302 (IF=1 SF=0 ZF=0 CF=0 OF=0)
Tutor> md 200                        //Check target area: nothing yet
00000200    00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
Tutor> t                             //Execute 2nd instruction
Exception 1 at EIP=00100108: Debugger interrupt
Tutor> rd                           //See b in EAX, EIP to offset 8
EAX=0000000b EBX=00009e00   EBP=000578ac
EDX=00101b88 ECX=00101bac   ESP=003ffff0
ESI=00090800 EDI=00101d5c   EIP=00100108
EFLAGS=0302 (IF=1 SF=0 ZF=0 CF=0 OF=0)
Tutor> md 200                        //Check target area: nothing yet
00000200    00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
Tutor> t                             //Execute 3rd instruction
Exception 1 at EIP=0010010d: Debugger interrupt
Tutor> rd                           //Only EIP has changed in regs
EAX=0000000b EBX=00009e00   EBP=000578ac
EDX=00101b88 ECX=00101bac   ESP=003ffff0
ESI=00090800 EDI=00101d5c   EIP=0010010d
EFLAGS=0302 (IF=1 SF=0 ZF=0 CF=0 OF=0)
Tutor> md 200                        //Check mem--yes, 0b now in 0x200
00000200    0b 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
Tutor> t                             //Execute int $3
Exception 3 at EIP=0010010e: Breakpoint
Tutor> ~q
Quit handler:
Killing process xxxx Leaving board #1
Tutor-vserver$
```

5. Try out remote gdb on tiny: See details in part 6 of <http://www.cs.umb.edu/~cheungr/cs341/VMWare-for-Tutor.pdf>. For the VM environment, COM1 is for remote gdb and COM2 is for the console.

```
-----
At the tutor-vserver VM, enter:
Tutor-vserver$ mtip -f tiny.lnx  (always use board #1)
  For command help, type ~?
  For help on args, rerun without args
  Code starts at 0x100100
  Using board # 1
  (hit <CR> here)

Tutor> ~d
  .Done.
  Download done, setting eip to 100100
Tutor> gdb
  Setting gdb dev to COM1, starting gdb (CTRL-C to abort).
  <---just let it hang here
-----
```

In another window in your PC, run putty. Connect putty to the tutor-vserver VM's IP address. Logon to tutor-vserver VM using the same credentials provided.

```
-----
Tutor-vserver$
Tutor-vserver$ gdb tiny.lnx
GNU gdb (GDB) 7.0.1-debian
Copyright (C) 2009 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show copying"
and "show warranty" for details.
This GDB was configured as "i486-linux-gnu".
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>...
Reading symbols from /home/tuser/cs341/mp2/warmup/tiny.lnx...(no debugging
symbols found)...done.
(gdb) tar rem /dev/ttyS0          <--set gdb to talk to COM1
Remote debugging using /dev/ttyS0
0x00100100 in ?? ()
(gdb) set $eip=0x100100          <--set PC to point at 0x100100
(gdb) i reg
eax                0xb                11
ecx                0x6a894             436372
edx                0x0                 0
ebx                0x9e00              40448
esp                0x578a8             0x578a8
ebp                0x578ac             0x578ac
esi                0x90800             591872
edi                0x51ffc             335868
eip                0x100100            0x100100
ps                 0x302               770
cs                 0x10                16
ss                 0x18                24
ds                 0x18                24
es                 0x18                24
fs                 0x18                24
-----
```

```

gs                0x18                24
(gdb) x/x 0x200
0x200:    0x00000abc    <--old contents of memory at 0x200
(gdb) set *(int *)0x200 = 0    <--how to "ms" with gdb
(gdb) x/x 0x200                <--check results
0x200:    0x00000000
(gdb) set $eip = 0x100100    <--to run from start
(gdb) x/4i 0x100100    <--examine 4 instructions
0x100100 <tiny.opc>:    movl    $0x8,%eax
0x100105 <tiny.opc+5>:    addl    $0x3,%eax
0x100108 <tiny.opc+8>:    movl    %eax,0x200
0x10010d <tiny.opc+13>:   int3
(gdb) b *0x100105    <--set breakpoint at 2nd instruction
Breakpoint 1 at 0x100105
(gdb) c                <--continue from 0x100100
Continuing.

```

Breakpoint 1, 0x100105 in tiny.opc ()

```

(gdb) i reg
eax                0x8                8
ecx                0x6a894            436372
edx                0x0                0
ebx                0x9e00            40448
esp                0x578a8            0x578a8
ebp                0x578ac            0x578ac
esi                0x90800            591872
edi                0x51ffc            335868
eip                0x100105            0x100105
ps                0x216                534
cs                0x10                16
ss                0x18                24
ds                0x18                24
es                0x18                24
fs                0x18                24
gs                0x18                24
(gdb) b *0x100108
Breakpoint 2 at 0x100108
(gdb) c
Continuing.

```

Breakpoint 2, 0x100108 in tiny.opc ()

```

(gdb) i reg
eax                0xb                11
ecx                0x6a894            436372
edx                0x0                0
ebx                0x9e00            40448
esp                0x578a8            0x578a8
ebp                0x578ac            0x578ac
esi                0x90800            591872
edi                0x51ffc            335868
eip                0x100108            0x100108
ps                0x202                514
cs                0x10                16
ss                0x18                24
ds                0x18                24
es                0x18                24
fs                0x18                24

```

```

gs                0x18                24

(gdb) b *0x10010d
Breakpoint 3 at 0x10010d
(gdb) c
Continuing.

Breakpoint 3, 0x10010d in tiny.opc ()

(gdb) i reg
eax                0xb                11
ecx                0x6a894            436372
edx                0x0                0
ebx                0x9e00            40448
esp                0x578a8            0x578a8
ebp                0x578ac            0x578ac
esi                0x90800            591872
edi                0x51ffc            335868
eip                0x10010d            0x10010d
ps                 0x302              770
cs                 0x10              16
ss                 0x18              24
ds                 0x18              24
es                 0x18              24
fs                 0x18              24
gs                 0x18              24
(gdb) x/x 0x200
0x200: 0x0000000b
(gdb) q
The program is running.  Quit anyway (and kill it)? (y or n) y
Tutor-vserver$

```

Note: To everyone who may encounter this problem and ask:

Question: Why I am I getting these error messages?

```
u18(9)% cat tiny.s
# tiny.s
# mp2 Warmup

    movl $8, %eax
    addl $3, %eax
    movl %eax, 0x200
    int $3
.end

u18(10)% i386-as -o tiny.opc tiny.s
tiny.s: Assembler messages:
tiny.s:4: Error: Rest of line ignored. First ignored character valued 0xd.
tiny.s:5: Error: invalid character (0xd) in second operand
tiny.s:6: Error: invalid character (0xd) in second operand
tiny.s:7: Error: invalid character (0xd) in second operand
tiny.s:8: Error: invalid character (0xd) in first operand
tiny.s:9: Error: Rest of line ignored. First ignored character valued 0xd.
```

Answer:

You must have used an editor such as notepad on your PC locally to create the .s file and used file transfer to put it on the ulab system. Notepad has put a carriage return (CR) character 0x0d at the end of each line in addition to the normal UNIX new line (NL/LF) character 0x0a.

Here is an octal dump of the ASCII characters in hex form that are in your source file:

```
u18(56)% od -x tiny.s
0000000 2320 7469 6e79 2e73 0d0a 2320 4761 6c69
0000020 6e61 204f 736d 6f6c 6f76 736b 6179 610d
0000040 0a23 206d 7032 2057 6172 6d75 700d 0a0a
0000060 2020 206d 6f76 6c20 2438 2c20 2565 6178
0000100 0a20 2020 6164 646c 2024 332c 2025 6561
0000120 780a 2020 206d 6f76 6c20 2565 6178 2c20
0000140 3078 3230 300a 2020 2069 6e74 2024 330a
0000160 2020 2e65 6e64 0a00 0000167
u18(57)%
```

Notice the 0d0a character sequence that occurs at the end of each line.

The GAS assembler(i386-as) is not ignoring the carriage return character 0x0d at the end of each line and it gives an error. To fix this problem, you can use an UNIX editor such as vi to remove the carriage return (CR) characters or you can use the UNIX command tr to remove the 0x0d (or octal 15) characters and the command mv to rename the output file to the original one:

```
u18(58)% tr -d '\015' <tiny.s >output_file
u18(58)% mv output_file tiny.s
```