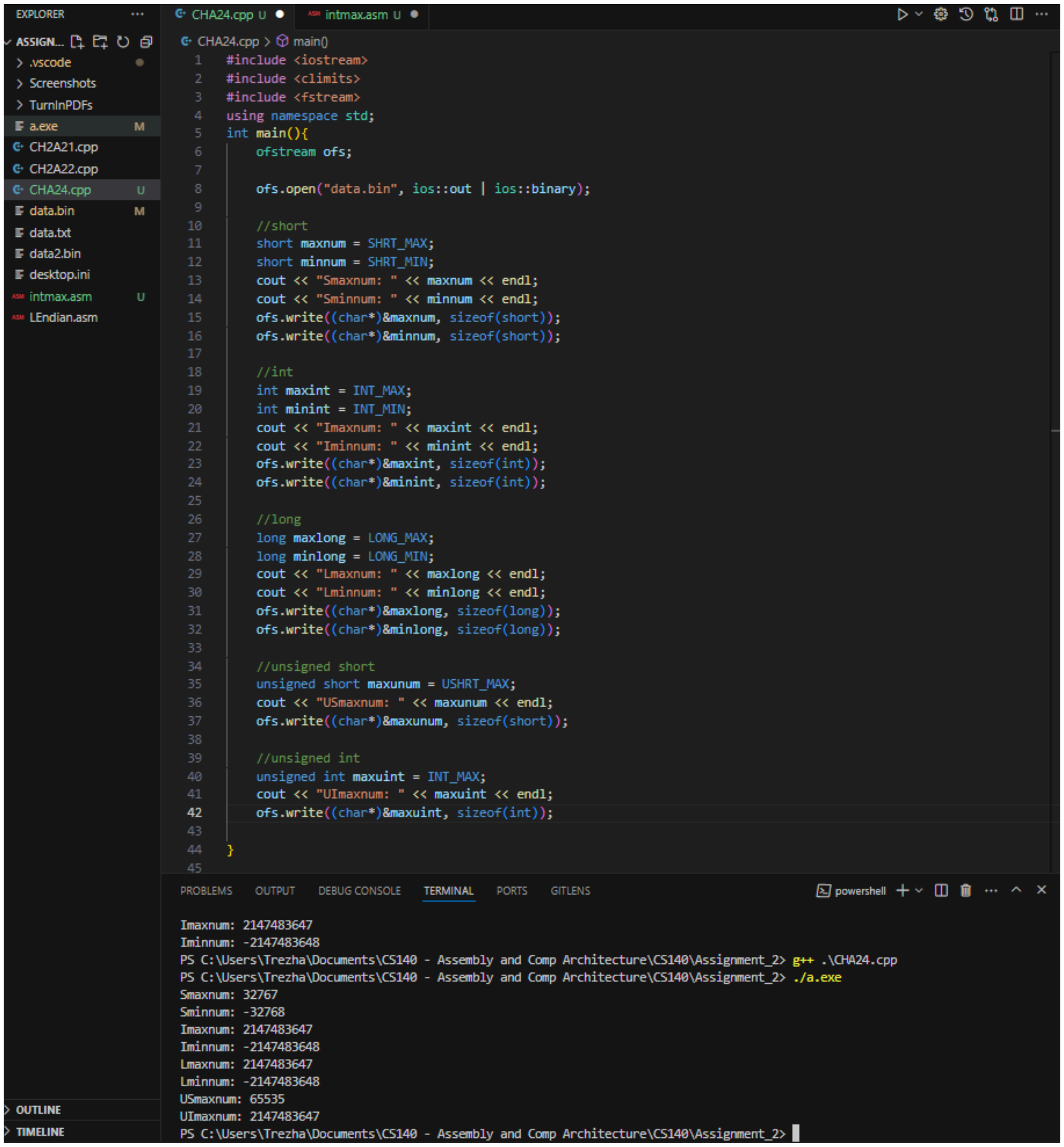
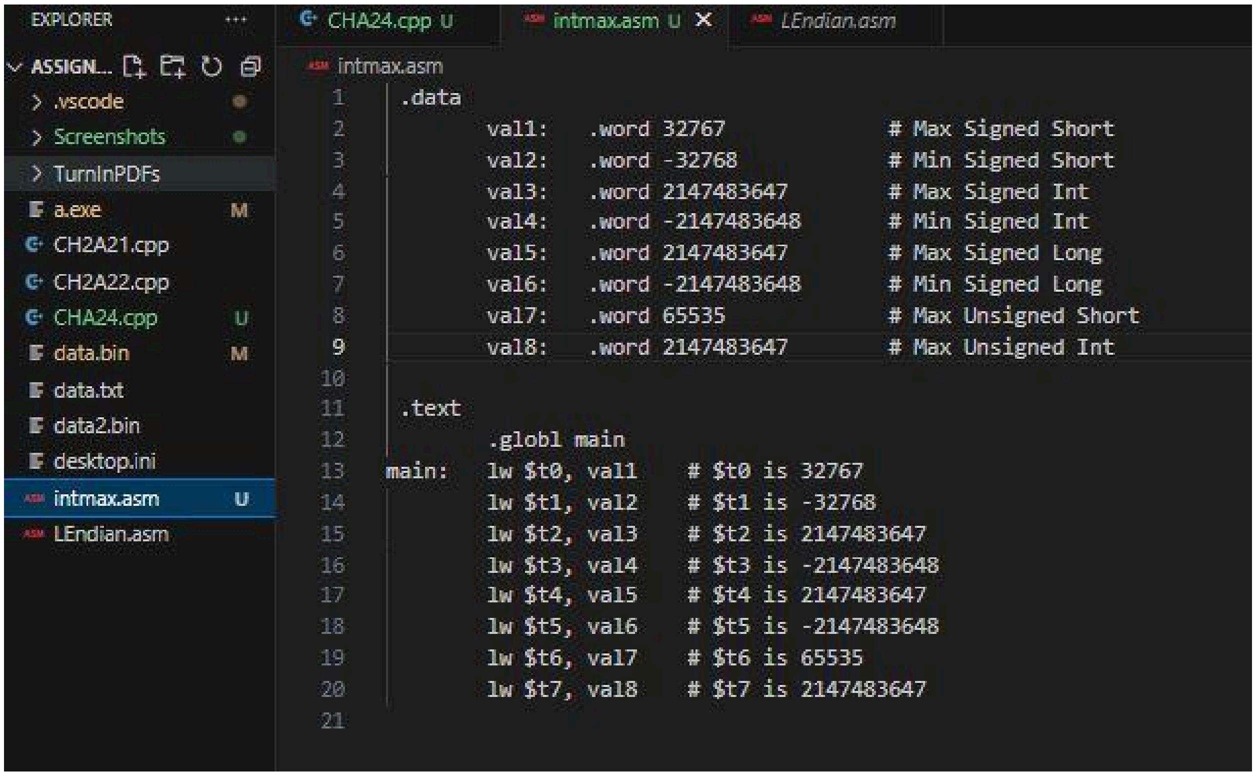
Assignment 2-4 MAX and MIN Value of Integer Data Types in Assembly Program

1. **Make a program to show those limit numbers.**
    a. **INT_MAX** got that **+ MIN**
    b. **SHRT_MAX** got that **+ MIN**
    c. **USHRT_MAX** got that, **no Minimum** because it's unassigned so the lowest will always be 0
    d. **UINT_MAX** got that, **no Minimum** for same reason above.
    e. **LONG_MAX** got that + MIN

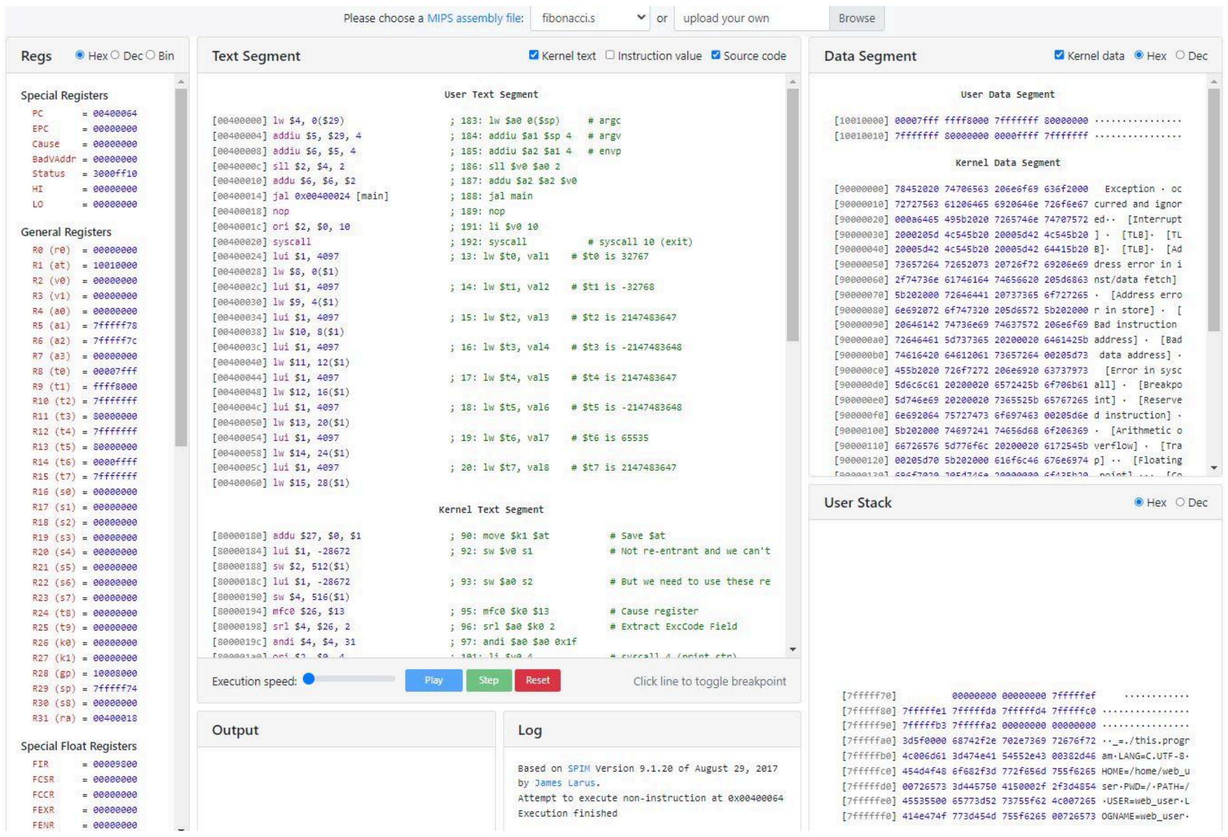


```cpp
#include <iostream>
#include <climits>
#include <fstream>
using namespace std;
int main(){
    ofstream ofs;

    ofs.open("data.bin", ios::out | ios::binary);

    //short
    short maxnum = SHRT_MAX;
    short minnum = SHRT_MIN;
    cout << "Smaxnum: " << maxnum << endl;
    cout << "Sminnum: " << minnum << endl;
    ofs.write((char*)&maxnum, sizeof(short));
    ofs.write((char*)&minnum, sizeof(short));

    //int
    int maxint = INT_MAX;
    int minint = INT_MIN;
    cout << "Imaxnum: " << maxint << endl;
    cout << "Iminnum: " << minint << endl;
    ofs.write((char*)&maxint, sizeof(int));
    ofs.write((char*)&minint, sizeof(int));

    //long
    long maxlong = LONG_MAX;
    long minlong = LONG_MIN;
    cout << "Lmaxnum: " << maxlong << endl;
    cout << "Lminnum: " << minlong << endl;
    ofs.write((char*)&maxlong, sizeof(long));
    ofs.write((char*)&minlong, sizeof(long));

    //unsigned short
    unsigned short maxunum = USHRT_MAX;
    cout << "USmaxnum: " << maxunum << endl;
    ofs.write((char*)&maxunum, sizeof(short));

    //unsigned int
    unsigned int maxuint = INT_MAX;
    cout << "UImaxnum: " << maxuint << endl;
    ofs.write((char*)&maxuint, sizeof(int));

}
```

```
Imaxnum: 2147483647
Iminnum: -2147483648
PS C:\Users\Trezha\Documents\CS140 - Assembly and Comp Architecture\CS140\Assignment_2> g++ .\CHA24.cpp
PS C:\Users\Trezha\Documents\CS140 - Assembly and Comp Architecture\CS140\Assignment_2> ./a.exe
Smaxnum: 32767
Sminnum: -32768
Imaxnum: 2147483647
Iminnum: -2147483648
Lmaxnum: 2147483647
Lminnum: -2147483648
USmaxnum: 65535
UImaxnum: 2147483647
PS C:\Users\Trezha\Documents\CS140 - Assembly and Comp Architecture\CS140\Assignment_2>
```



```
.data
        val1:   .word 32767             # Max Signed Short
        val2:   .word -32768            # Min Signed Short
        val3:   .word 2147483647        # Max Signed Int
        val4:   .word -2147483648       # Min Signed Int
        val5:   .word 2147483647        # Max Signed Long
        val6:   .word -2147483648       # Min Signed Long
        val7:   .word 65535             # Max Unsigned Short
        val8:   .word 2147483647        # Max Unsigned Int

.text
        .globl main
main:   lw $t0, val1    # $t0 is 32767
        lw $t1, val2    # $t1 is -32768
        lw $t2, val3    # $t2 is 2147483647
        lw $t3, val4    # $t3 is -2147483648
        lw $t4, val5    # $t4 is 2147483647
        lw $t5, val6    # $t5 is -2147483648
        lw $t6, val7    # $t6 is 65535
        lw $t7, val8    # $t7 is 2147483647
```


Please choose a MIPS assembly file: fibonacci.s
or upload your own
Browse
Regs [✓ Hex] Dec Bin
Text Segment [✓ Kernel text] Instruction value [✓ Source code]
Data Segment [✓ Kernel data] [✓ Hex] Dec
User Stack [✓ Hex] Dec
Execution speed:
Play
Step
Reset
Click line to toggle breakpoint
Output
Log

```
Special Registers
  PC       = 00400064
  EPC      = 00000000
  Cause    = 00000000
  BadVAddr = 00000000
  Status   = 3000ff10
  HI       = 00000000
  LO       = 00000000

General Registers
  R0 (r0)  = 00000000
  R1 (at)  = 10010000
  R2 (v0)  = 00000000
  R3 (v1)  = 00000000
  R4 (a0)  = 00000000
  R5 (a1)  = 7ffff78
  R6 (a2)  = 7ffff7c
  R7 (a3)  = 00000000
  R8 (t0)  = 00007fff
  R9 (t1)  = ffff8000
  R10 (t2) = 7fffffff
  R11 (t3) = 80000000
  R12 (t4) = 7fffffff
  R13 (t5) = 80000000
  R14 (t6) = 0000ffff
  R15 (t7) = 7fffffff
  R16 (s0) = 00000000
  R17 (s1) = 00000000
  R18 (s2) = 00000000
  R19 (s3) = 00000000
  R20 (s4) = 00000000
  R21 (s5) = 00000000
  R22 (s6) = 00000000
  R23 (s7) = 00000000
  R24 (t8) = 00000000
  R25 (t9) = 00000000
  R26 (k0) = 00000000
  R27 (k1) = 00000000
  R28 (gp) = 10008000
  R29 (sp) = 7ffff74
  R30 (s8) = 00000000
  R31 (ra) = 00400018

Special Float Registers
  FIR      = 00009800
  FCSR     = 00000000
  FCCR     = 00000000
  FEXR     = 00000000
  FENR     = 00000000
```

```
                              User Text Segment

[00400000] lw $4, 0($29)              ; 183: lw $a0 0($sp)      # argc
[00400004] addiu $5, $29, 4           ; 184: addiu $a1 $sp 4    # argv
[00400008] addiu $6, $5, 4            ; 185: addiu $a2 $a1 4    # envp
[0040000c] sll $2, $4, 2              ; 186: sll $v0 $a0 2
[00400010] addu $6, $6, $2            ; 187: addu $a2 $a2 $v0
[00400014] jal 0x00400024 [main]      ; 188: jal main
[00400018] nop                        ; 189: nop
[0040001c] ori $2, $0, 10             ; 191: li $v0 10
[00400020] syscall                    ; 192: syscall            # syscall 10 (exit)
[00400024] lui $1, 4097               ; 13: lw $t0, val1    # $t0 is 32767
[00400028] lw $8, 0($1)
[0040002c] lui $1, 4097               ; 14: lw $t1, val2    # $t1 is -32768
[00400030] lw $9, 4($1)
[00400034] lui $1, 4097               ; 15: lw $t2, val3    # $t2 is 2147483647
[00400038] lw $10, 8($1)
[0040003c] lui $1, 4097               ; 16: lw $t3, val4    # $t3 is -2147483648
[00400040] lw $11, 12($1)
[00400044] lui $1, 4097               ; 17: lw $t4, val5    # $t4 is 2147483647
[00400048] lw $12, 16($1)
[0040004c] lui $1, 4097               ; 18: lw $t5, val6    # $t5 is -2147483648
[00400050] lw $13, 20($1)
[00400054] lui $1, 4097               ; 19: lw $t6, val7    # $t6 is 65535
[00400058] lw $14, 24($1)
[0040005c] lui $1, 4097               ; 20: lw $t7, val8    # $t7 is 2147483647
[00400060] lw $15, 28($1)

                              Kernel Text Segment

[80000180] addu $27, $0, $1           ; 90: move $k1 $at        # Save $at
[80000184] lui $1, -28672             ; 92: sw $v0 s1           # Not re-entrant and we can't
[80000188] sw $2, 512($1)
[8000018c] lui $1, -28672             ; 93: sw $a0 s2           # But we need to use these re
[80000190] sw $4, 516($1)
[80000194] mfc0 $26, $13              ; 95: mfc0 $k0 $13        # Cause register
[80000198] srl $4, $26, 2             ; 96: srl $a0 $k0 2       # Extract ExcCode Field
[8000019c] andi $4, $4, 31            ; 97: andi $a0 $a0 0x1f
```

```
                         User Data Segment

[10010000] 00007fff ffff8000 7fffffff 80000000 ................
[10010010] 7fffffff 80000000 0000ffff 7fffffff ................

                        Kernel Data Segment

[90000000] 78452020 74706563 206e6f69 636f2000   Exception · oc
[90000010] 72727563 61206465 6920646e 726f6e67 curred and ignor
[90000020] 000a6465 495b2020 7265746e 74707572 ed··  [Interrupt
[90000030] 2000205d 4c545b20 20005d42 4c545b20 ] ·  [TLB]·  [TL
[90000040] 20005d42 4c545b20 20005d42 64415b20 B]·  [TLB]·  [Ad
[90000050] 73657264 72652073 20726f72 69206e69 dress error in i
[90000060] 2f74736e 61746164 74656620 205d6863 nst/data fetch]
[90000070] 5b202000 72646441 20737365 6f727265 ·  [Address erro
[90000080] 6e692072 6f747320 205d6572 5b202000 r in store] ·  [
[90000090] 20646142 74736e69 74637572 206e6f69 Bad instruction
[900000a0] 72646461 5d737365 20200020 6461425b address] ·  [Bad
[900000b0] 74616420 64612061 73657264 00205d73  data address] ·
[900000c0] 455b2020 726f7272 206e6920 63737973   [Error in sysc
[900000d0] 5d6c6c61 20200020 6572425b 6f706b61 all] ·  [Breakpo
[900000e0] 5d746e69 20200020 7365525b 65767265 int] ·  [Reserve
[900000f0] 6e692064 75727473 6f697463 00205d6e d instruction] ·
[90000100] 5b202000 74697241 74656d68 6f206369 ·  [Arithmetic o
[90000110] 66726576 5d776f6c 20200020 6172545b verflow] ·  [Tra
[90000120] 00205d70 5b202000 616f6c46 676e6974 p] ··  [Floating
```

```
[7ffff70]          00000000 00000000 7ffffef     ............
[7ffff80] 7ffffe1 7ffffda 7ffffd4 7ffffc0 ................
[7ffff90] 7ffffb3 7ffffa2 00000000 00000000 ................
[7ffffa0] 3d5f0000 68742f2e 702e7369 72676f72 ··_=./this.progr
[7ffffb0] 4c006d61 3d474e41 54552e43 00382d46 am·LANG=C.UTF-8·
[7ffffc0] 454d4f48 6f682f3d 772f656d 755f6265 HOME=/home/web_u
[7ffffd0] 00726573 3d445750 4150002f 2f3d4854 ser·PWD=/·PATH=/
[7ffffe0] 45535500 65773d52 73755f62 4c007265 ·USER=web_user·L
[7fffff0] 414e474f 773d454d 755f6265 00726573 OGNAME=web_user·
```

```
Based on SPIM Version 9.1.20 of August 29, 2017
by James Larus.
Attempt to execute non-instruction at 0x00400064
Execution finished
```

2. **Elaborate on the limit numbers based on 2's complement number systems**
    a. **Why 32767 for short type? Prove it.**
    A short consists of 16 bits. In a 2's complementary system, 1 bit is used for the sign while the remaining 15 bits are used for magnitude. Therefore the range of a short is **-2^15 to 2^15 -1.**
    We can prove this using the same equation, **2^15 => 32768 -1 = 32767(00007fff)**
    -2^15 = -32768(ffff8000)

    For Unsigned Shorts, we won't include the negative range because the minimum for unsigned numbers are 0. So the range is only 0 to 2^15 -1.

    b. **Elaborate on other limit numbers**
    Int's consists of 32 bits. So in a 2's complementary system, 1 bit is used for the sign while the remaining 31 bits are used for magnitude. Therefore the range of an int is -**2^31 to 2^31 -1.**
    We can prove this using the equation **2^31= 2147483648 - 1 =**
    2147483647(7fffffff)
    -2^31= -2147483648(80000000)

    Similar to the above question, Unsigned Ints minimum value is 0, so it's range is only from 0 to 2^31 -1.

    For **Long data types**, they too hold 32 bits similar to the Int mentioned above, so because of this reason, **they have the same exact range as signed Int's**.

    (Reminder that we're using the c++ macro's <climits> to define what the min and max values of long are, and from my end it outputs the same value as int (as seen in the first screenshot.))