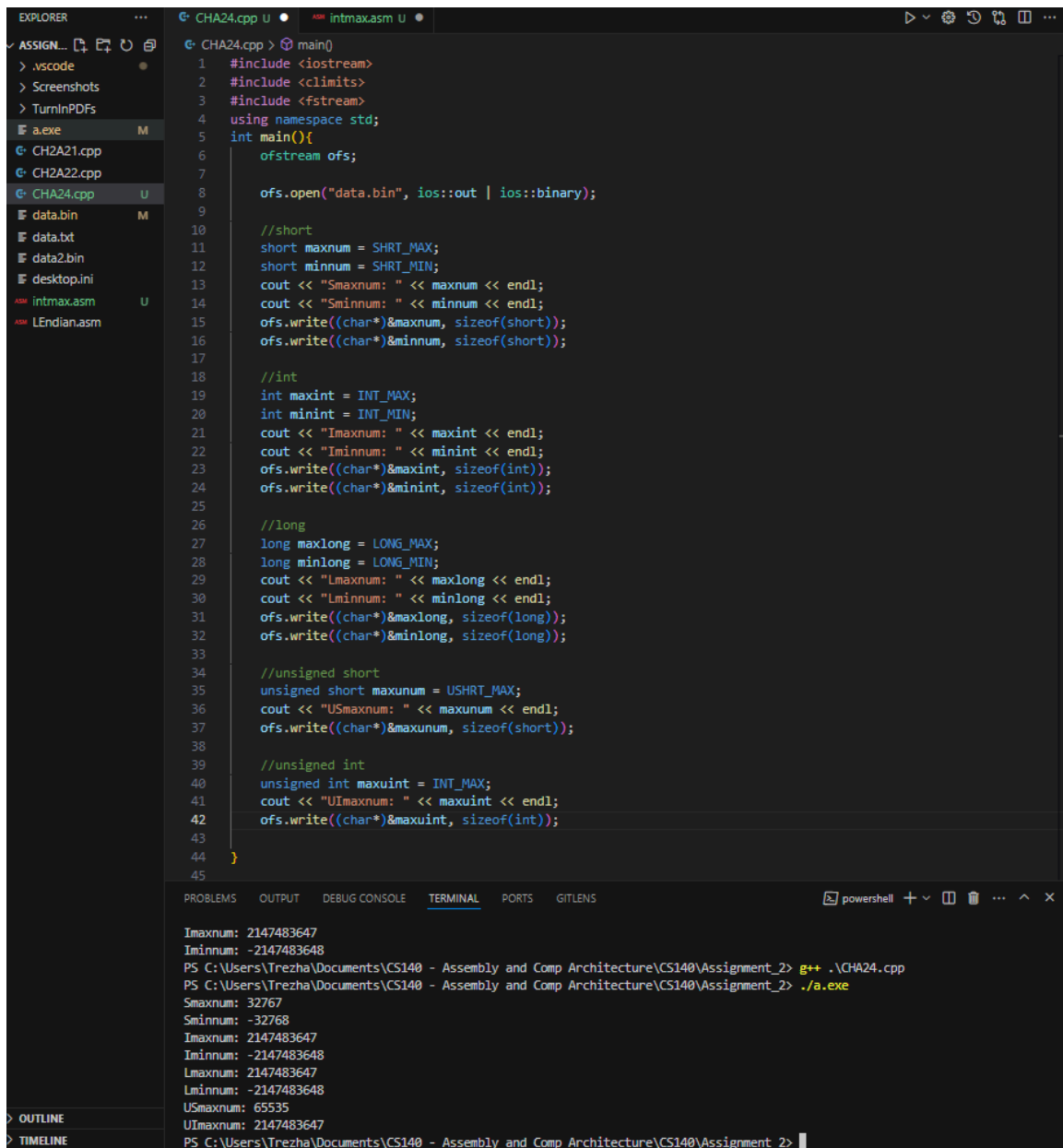


Assignment 2-4 MAX and MIN Value of Integer Data Types in Assembly Program

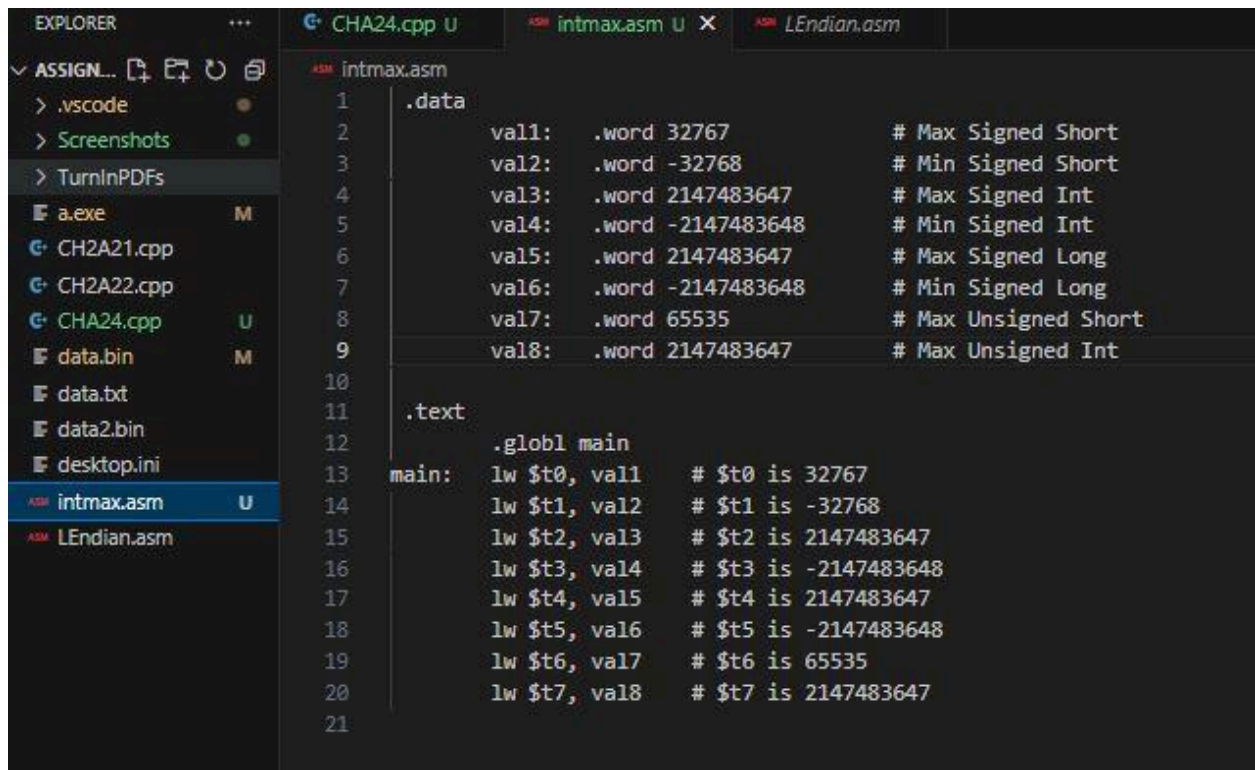
1. Make a program to show those limit numbers.
 - a. **INT_MAX** got that + **MIN**
 - b. **SHRT_MAX** got that + **MIN**
 - c. **USHRT_MAX** got that, **no Minimum** because it's unassigned so the lowest will always be 0
 - d. **UINT_MAX** got that, **no Minimum** for same reason above.
 - e. **LONG_MAX**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



The screenshot shows a Visual Studio Code editor with a C++ program in `CHA24.cpp` and its output in the terminal. The program calculates and prints the maximum and minimum values for various integer data types. The output shows the following values:

```
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>
```

```
1  #include <iostream>
2  #include <climits>
3  #include <fstream>
4  using namespace std;
5  int main(){
6      ofstream ofs;
7
8      ofs.open("data.bin", ios::out | ios::binary);
9
10     //short
11     short maxnum = SHRT_MAX;
12     short minnum = SHRT_MIN;
13     cout << "Smaxnum: " << maxnum << endl;
14     cout << "Sminnum: " << minnum << endl;
15     ofs.write((char*)&maxnum, sizeof(short));
16     ofs.write((char*)&minnum, sizeof(short));
17
18     //int
19     int maxint = INT_MAX;
20     int minint = INT_MIN;
21     cout << "Imaxnum: " << maxint << endl;
22     cout << "Iminnum: " << minint << endl;
23     ofs.write((char*)&maxint, sizeof(int));
24     ofs.write((char*)&minint, sizeof(int));
25
26     //long
27     long maxlong = LONG_MAX;
28     long minlong = LONG_MIN;
29     cout << "Lmaxnum: " << maxlong << endl;
30     cout << "Lminnum: " << minlong << endl;
31     ofs.write((char*)&maxlong, sizeof(long));
32     ofs.write((char*)&minlong, sizeof(long));
33
34     //unsigned short
35     unsigned short maxunum = USHRT_MAX;
36     cout << "USmaxnum: " << maxunum << endl;
37     ofs.write((char*)&maxunum, sizeof(short));
38
39     //unsigned int
40     unsigned int maxuint = UINT_MAX;
41     cout << "UImaxnum: " << maxuint << endl;
42     ofs.write((char*)&maxuint, sizeof(int));
43
44 }
```



Please choose a MIPS assembly file: fibonacci.s or upload your own Browse

Regs Hex Dec Bin

Special Registers

PC	= 00400004
EPC	= 00000000
Cause	= 00000000
BadVAddr	= 00000000
Status	= 3000ff10
HI	= 00000000
LO	= 00000000

General Registers

R0 (\$r0)	= 00000000
R1 (\$t0)	= 00000000
R2 (\$t1)	= 00000000
R3 (\$t2)	= 00000000
R4 (\$t3)	= 00000000
R5 (\$t4)	= 00000000
R6 (\$t5)	= 00000000
R7 (\$t6)	= 00000000
R8 (\$t7)	= 00000000
R9 (\$f0)	= 00000000
R10 (\$f1)	= 00000000
R11 (\$f2)	= 00000000
R12 (\$f3)	= 00000000
R13 (\$f4)	= 00000000
R14 (\$f5)	= 00000000
R15 (\$f6)	= 00000000
R16 (\$f7)	= 00000000
R17 (\$f8)	= 00000000
R18 (\$f9)	= 00000000
R19 (\$f10)	= 00000000
R20 (\$f11)	= 00000000
R21 (\$f12)	= 00000000
R22 (\$f13)	= 00000000
R23 (\$f14)	= 00000000
R24 (\$f15)	= 00000000
R25 (\$f16)	= 00000000
R26 (\$f17)	= 00000000
R27 (\$f18)	= 00000000
R28 (\$f19)	= 00000000
R29 (\$f20)	= 00000000
R30 (\$f21)	= 00000000
R31 (\$f22)	= 00000000

Special Float Registers

FIR	= 00000000
FCR0	= 00000000
FCR1	= 00000000
FCR2	= 00000000
FCR3	= 00000000
FCR4	= 00000000
FCR5	= 00000000
FCR6	= 00000000
FCR7	= 00000000
FCR8	= 00000000
FCR9	= 00000000
FCR10	= 00000000
FCR11	= 00000000
FCR12	= 00000000
FCR13	= 00000000
FCR14	= 00000000
FCR15	= 00000000
FCR16	= 00000000
FCR17	= 00000000
FCR18	= 00000000
FCR19	= 00000000
FCR20	= 00000000
FCR21	= 00000000
FCR22	= 00000000
FCR23	= 00000000
FCR24	= 00000000
FCR25	= 00000000
FCR26	= 00000000
FCR27	= 00000000
FCR28	= 00000000
FCR29	= 00000000
FCR30	= 00000000
FCR31	= 00000000

Text Segment Kernel text Instruction value Source code

User Text Segment

```
[00400000] lw $t0, 0($t0) ; 183: lw $t0, 0($t0) # argc
[00400004] addiu $t1, $t0, 4 ; 184: addiu $t1, $t0, 4 # argv
[00400008] addiu $t2, $t1, 4 ; 185: addiu $t2, $t1, 4 # envp
[0040000c] sll $t3, $t2, 2 ; 186: sll $t3, $t2, 2
[00400010] addu $t4, $t2, $t3 ; 187: addu $t4, $t2, $t3
[00400014] jal 0x00400024 [main] ; 188: jal main
[00400018] nop ; 189: nop
[0040001c] ori $t5, $0, 10 ; 191: li $t5, 10
[00400020] syscall ; 192: syscall # syscall 10 (exit)
[00400024] lui $t6, 0x1 ; 193: lw $t6, val1 # $t6 is 32767
[00400028] lui $t7, 0x1 ; 194: lw $t7, val2 # $t7 is -32768
[0040002c] lui $t8, 0x1 ; 195: lw $t8, val3 # $t8 is 2147483647
[00400030] lui $t9, 0x1 ; 196: lw $t9, val4 # $t9 is -2147483648
[00400034] lui $t10, 0x1 ; 197: lw $t10, val5 # $t10 is 2147483647
[00400038] lui $t11, 0x1 ; 198: lw $t11, val6 # $t11 is -2147483648
[0040003c] lui $t12, 0x1 ; 199: lw $t12, val7 # $t12 is 65535
[00400040] lui $t13, 0x1 ; 200: lw $t13, val8 # $t13 is 2147483647
[00400044] lui $t14, 0x1 ; 201: lw $t14, val8 # $t14 is 2147483647
[00400048] lui $t15, 0x1 ; 202: lw $t15, val8 # $t15 is 2147483647
[0040004c] lui $t16, 0x1 ; 203: lw $t16, val8 # $t16 is 2147483647
[00400050] lui $t17, 0x1 ; 204: lw $t17, val8 # $t17 is 2147483647
[00400054] lui $t18, 0x1 ; 205: lw $t18, val8 # $t18 is 2147483647
[00400058] lui $t19, 0x1 ; 206: lw $t19, val8 # $t19 is 2147483647
[0040005c] lui $t20, 0x1 ; 207: lw $t20, val8 # $t20 is 2147483647
[00400060] lui $t21, 0x1 ; 208: lw $t21, val8 # $t21 is 2147483647
[00400064] lui $t22, 0x1 ; 209: lw $t22, val8 # $t22 is 2147483647
[00400068] lui $t23, 0x1 ; 210: lw $t23, val8 # $t23 is 2147483647
[0040006c] lui $t24, 0x1 ; 211: lw $t24, val8 # $t24 is 2147483647
[00400070] lui $t25, 0x1 ; 212: lw $t25, val8 # $t25 is 2147483647
[00400074] lui $t26, 0x1 ; 213: lw $t26, val8 # $t26 is 2147483647
[00400078] lui $t27, 0x1 ; 214: lw $t27, val8 # $t27 is 2147483647
[0040007c] lui $t28, 0x1 ; 215: lw $t28, val8 # $t28 is 2147483647
[00400080] lui $t29, 0x1 ; 216: lw $t29, val8 # $t29 is 2147483647
[00400084] lui $t30, 0x1 ; 217: lw $t30, val8 # $t30 is 2147483647
[00400088] lui $t31, 0x1 ; 218: lw $t31, val8 # $t31 is 2147483647
[0040008c] lui $t32, 0x1 ; 219: lw $t32, val8 # $t32 is 2147483647
[00400090] lui $t33, 0x1 ; 220: lw $t33, val8 # $t33 is 2147483647
[00400094] lui $t34, 0x1 ; 221: lw $t34, val8 # $t34 is 2147483647
[00400098] lui $t35, 0x1 ; 222: lw $t35, val8 # $t35 is 2147483647
[0040009c] lui $t36, 0x1 ; 223: lw $t36, val8 # $t36 is 2147483647
[004000a0] lui $t37, 0x1 ; 224: lw $t37, val8 # $t37 is 2147483647
[004000a4] lui $t38, 0x1 ; 225: lw $t38, val8 # $t38 is 2147483647
[004000a8] lui $t39, 0x1 ; 226: lw $t39, val8 # $t39 is 2147483647
[004000ac] lui $t40, 0x1 ; 227: lw $t40, val8 # $t40 is 2147483647
[004000b0] lui $t41, 0x1 ; 228: lw $t41, val8 # $t41 is 2147483647
[004000b4] lui $t42, 0x1 ; 229: lw $t42, val8 # $t42 is 2147483647
[004000b8] lui $t43, 0x1 ; 230: lw $t43, val8 # $t43 is 2147483647
[004000bc] lui $t44, 0x1 ; 231: lw $t44, val8 # $t44 is 2147483647
[004000c0] lui $t45, 0x1 ; 232: lw $t45, val8 # $t45 is 2147483647
[004000c4] lui $t46, 0x1 ; 233: lw $t46, val8 # $t46 is 2147483647
[004000c8] lui $t47, 0x1 ; 234: lw $t47, val8 # $t47 is 2147483647
[004000cc] lui $t48, 0x1 ; 235: lw $t48, val8 # $t48 is 2147483647
[004000d0] lui $t49, 0x1 ; 236: lw $t49, val8 # $t49 is 2147483647
[004000d4] lui $t50, 0x1 ; 237: lw $t50, val8 # $t50 is 2147483647
[004000d8] lui $t51, 0x1 ; 238: lw $t51, val8 # $t51 is 2147483647
[004000dc] lui $t52, 0x1 ; 239: lw $t52, val8 # $t52 is 2147483647
[004000e0] lui $t53, 0x1 ; 240: lw $t53, val8 # $t53 is 2147483647
[004000e4] lui $t54, 0x1 ; 241: lw $t54, val8 # $t54 is 2147483647
[004000e8] lui $t55, 0x1 ; 242: lw $t55, val8 # $t55 is 2147483647
[004000ec] lui $t56, 0x1 ; 243: lw $t56, val8 # $t56 is 2147483647
[004000f0] lui $t57, 0x1 ; 244: lw $t57, val8 # $t57 is 2147483647
[004000f4] lui $t58, 0x1 ; 245: lw $t58, val8 # $t58 is 2147483647
[004000f8] lui $t59, 0x1 ; 246: lw $t59, val8 # $t59 is 2147483647
[004000fc] lui $t60, 0x1 ; 247: lw $t60, val8 # $t60 is 2147483647
[00400100] lui $t61, 0x1 ; 248: lw $t61, val8 # $t61 is 2147483647
[00400104] lui $t62, 0x1 ; 249: lw $t62, val8 # $t62 is 2147483647
[00400108] lui $t63, 0x1 ; 250: lw $t63, val8 # $t63 is 2147483647
[0040010c] lui $t64, 0x1 ; 251: lw $t64, val8 # $t64 is 2147483647
[00400110] lui $t65, 0x1 ; 252: lw $t65, val8 # $t65 is 2147483647
[00400114] lui $t66, 0x1 ; 253: lw $t66, val8 # $t66 is 2147483647
[00400118] lui $t67, 0x1 ; 254: lw $t67, val8 # $t67 is 2147483647
[0040011c] lui $t68, 0x1 ; 255: lw $t68, val8 # $t68 is 2147483647
[00400120] lui $t69, 0x1 ; 256: lw $t69, val8 # $t69 is 2147483647
[00400124] lui $t70, 0x1 ; 257: lw $t70, val8 # $t70 is 2147483647
[00400128] lui $t71, 0x1 ; 258: lw $t71, val8 # $t71 is 2147483647
[0040012c] lui $t72, 0x1 ; 259: lw $t72, val8 # $t72 is 2147483647
[00400130] lui $t73, 0x1 ; 260: lw $t73, val8 # $t73 is 2147483647
[00400134] lui $t74, 0x1 ; 261: lw $t74, val8 # $t74 is 2147483647
[00400138] lui $t75, 0x1 ; 262: lw $t75, val8 # $t75 is 2147483647
[0040013c] lui $t76, 0x1 ; 263: lw $t76, val8 # $t76 is 2147483647
[00400140] lui $t77, 0x1 ; 264: lw $t77, val8 # $t77 is 2147483647
[00400144] lui $t78, 0x1 ; 265: lw $t78, val8 # $t78 is 2147483647
[00400148] lui $t79, 0x1 ; 266: lw $t79, val8 # $t79 is 2147483647
[0040014c] lui $t80, 0x1 ; 267: lw $t80, val8 # $t80 is 2147483647
[00400150] lui $t81, 0x1 ; 268: lw $t81, val8 # $t81 is 2147483647
[00400154] lui $t82, 0x1 ; 269: lw $t82, val8 # $t82 is 2147483647
[00400158] lui $t83, 0x1 ; 270: lw $t83, val8 # $t83 is 2147483647
[0040015c] lui $t84, 0x1 ; 271: lw $t84, val8 # $t84 is 2147483647
[00400160] lui $t85, 0x1 ; 272: lw $t85, val8 # $t85 is 2147483647
[00400164] lui $t86, 0x1 ; 273: lw $t86, val8 # $t86 is 2147483647
[00400168] lui $t87, 0x1 ; 274: lw $t87, val8 # $t87 is 2147483647
[0040016c] lui $t88, 0x1 ; 275: lw $t88, val8 # $t88 is 2147483647
[00400170] lui $t89, 0x1 ; 276: lw $t89, val8 # $t89 is 2147483647
[00400174] lui $t90, 0x1 ; 277: lw $t90, val8 # $t90 is 2147483647
[00400178] lui $t91, 0x1 ; 278: lw $t91, val8 # $t91 is 2147483647
[0040017c] lui $t92, 0x1 ; 279: lw $t92, val8 # $t92 is 2147483647
[00400180] lui $t93, 0x1 ; 280: lw $t93, val8 # $t93 is 2147483647
[00400184] lui $t94, 0x1 ; 281: lw $t94, val8 # $t94 is 2147483647
[00400188] lui $t95, 0x1 ; 282: lw $t95, val8 # $t95 is 2147483647
[0040018c] lui $t96, 0x1 ; 283: lw $t96, val8 # $t96 is 2147483647
[00400190] lui $t97, 0x1 ; 284: lw $t97, val8 # $t97 is 2147483647
[00400194] lui $t98, 0x1 ; 285: lw $t98, val8 # $t98 is 2147483647
[00400198] lui $t99, 0x1 ; 286: lw $t99, val8 # $t99 is 2147483647
[0040019c] lui $t100, 0x1 ; 287: lw $t100, val8 # $t100 is 2147483647
[004001a0] lui $t101, 0x1 ; 288: lw $t101, val8 # $t101 is 2147483647
[004001a4] lui $t102, 0x1 ; 289: lw $t102, val8 # $t102 is 2147483647
[004001a8] lui $t103, 0x1 ; 290: lw $t103, val8 # $t103 is 2147483647
[004001ac] lui $t104, 0x1 ; 291: lw $t104, val8 # $t104 is 2147483647
[004001b0] lui $t105, 0x1 ; 292: lw $t105, val8 # $t105 is 2147483647
[004001b4] lui $t106, 0x1 ; 293: lw $t106, val8 # $t106 is 2147483647
[004001b8] lui $t107, 0x1 ; 294: lw $t107, val8 # $t107 is 2147483647
[004001bc] lui $t108, 0x1 ; 295: lw $t108, val8 # $t108 is 2147483647
[004001c0] lui $t109, 0x1 ; 296: lw $t109, val8 # $t109 is 2147483647
[004001c4] lui $t110, 0x1 ; 297: lw $t110, val8 # $t110 is 2147483647
[004001c8] lui $t111, 0x1 ; 298: lw $t111, val8 # $t111 is 2147483647
[004001cc] lui $t112, 0x1 ; 299: lw $t112, val8 # $t112 is 2147483647
[004001d0] lui $t113, 0x1 ; 300: lw $t113, val8 # $t113 is 2147483647
[004001d4] lui $t114, 0x1 ; 301: lw $t114, val8 # $t114 is 2147483647
[004001d8] lui $t115, 0x1 ; 302: lw $t115, val8 # $t115 is 2147483647
[004001dc] lui $t116, 0x1 ; 303: lw $t116, val8 # $t116 is 2147483647
[004001e0] lui $t117, 0x1 ; 304: lw $t117, val8 # $t117 is 2147483647
[004001e4] lui $t118, 0x1 ; 305: lw $t118, val8 # $t118 is 2147483647
[004001e8] lui $t119, 0x1 ; 306: lw $t119, val8 # $t119 is 2147483647
[004001ec] lui $t120, 0x1 ; 307: lw $t120, val8 # $t120 is 2147483647
[004001f0] lui $t121, 0x1 ; 308: lw $t121, val8 # $t121 is 2147483647
[004001f4] lui $t122, 0x1 ; 309: lw $t122, val8 # $t122 is 2147483647
[004001f8] lui $t123, 0x1 ; 310: lw $t123, val8 # $t123 is 2147483647
[004001fc] lui $t124, 0x1 ; 311: lw $t124, val8 # $t124 is 2147483647
[00400200] lui $t125, 0x1 ; 312: lw $t125, val8 # $t125 is 2147483647
[00400204] lui $t126, 0x1 ; 313: lw $t126, val8 # $t126 is 2147483647
[00400208] lui $t127, 0x1 ; 314: lw $t127, val8 # $t127 is 2147483647
[0040020c] lui $t128, 0x1 ; 315: lw $t128, val8 # $t128 is 2147483647
[00400210] lui $t129, 0x1 ; 316: lw $t129, val8 # $t129 is 2147483647
[00400214] lui $t130, 0x1 ; 317: lw $t130, val8 # $t130 is 2147483647
[00400218] lui $t131, 0x1 ; 318: lw $t131, val8 # $t131 is 2147483647
[0040021c] lui $t132, 0x1 ; 319: lw $t132, val8 # $t132 is 2147483647
[00400220] lui $t133, 0x1 ; 320: lw $t133, val8 # $t133 is 2147483647
[00400224] lui $t134, 0x1 ; 321: lw $t134, val8 # $t134 is 2147483647
[00400228] lui $t135, 0x1 ; 322: lw $t135, val8 # $t135 is 2147483647
[0040022c] lui $t136, 0x1 ; 323: lw $t136, val8 # $t136 is 2147483647
[00400230] lui $t137, 0x1 ; 324: lw $t137, val8 # $t137 is 2147483647
[00400234] lui $t138, 0x1 ; 325: lw $t138, val8 # $t138 is 2147483647
[00400238] lui $t139, 0x1 ; 326: lw $t139, val8 # $t139 is 2147483647
[0040023c] lui $t140, 0x1 ; 327: lw $t140, val8 # $t140 is 2147483647
[00400240] lui $t141, 0x1 ; 328: lw $t141, val8 # $t141 is 2147483647
[00400244] lui $t142, 0x1 ; 329: lw $t142, val8 # $t142 is 2147483647
[00400248] lui $t143, 0x1 ; 330: lw $t143, val8 # $t143 is 2147483647
[0040024c] lui $t144, 0x1 ; 331: lw $t144, val8 # $t144 is 2147483647
[00400250] lui $t145, 0x1 ; 332: lw $t145, val8 # $t145 is 2147483647
[00400254] lui $t146, 0x1 ; 333: lw $t146, val8 # $t146 is 2147483647
[00400258] lui $t147, 0x1 ; 334: lw $t147, val8 # $t147 is 2147483647
[0040025c] lui $t148, 0x1 ; 335: lw $t148, val8 # $t148 is 2147483647
[00400260] lui $t149, 0x1 ; 336: lw $t149, val8 # $t149 is 2147483647
[00400264] lui $t150, 0x1 ; 337: lw $t150, val8 # $t150 is 2147483647
[00400268] lui $t151, 0x1 ; 338: lw $t151, val8 # $t151 is 2147483647
[0040026c] lui $t152, 0x1 ; 339: lw $t152, val8 # $t152 is 2147483647
[00400270] lui $t153, 0x1 ; 340: lw $t153, val8 # $t153 is 2147483647
[00400274] lui $t154, 0x1 ; 341: lw $t154, val8 # $t154 is 2147483647
[00400278] lui $t155, 0x1 ; 342: lw $t155, val8 # $t155 is 2147483647
[0040027c] lui $t156, 0x1 ; 343: lw $t156, val8 # $t156 is 2147483647
[00400280] lui $t157, 0x1 ; 344: lw $t157, val8 # $t157 is 2147483647
[00400284] lui $t158, 0x1 ; 345: lw $t158, val8 # $t158 is 2147483647
[00400288] lui $t159, 0x1 ; 346: lw $t159, val8 # $t159 is 2147483647
[0040028c] lui $t160, 0x1 ; 347: lw $t160, val8 # $t160 is 2147483647
[00400290] lui $t161, 0x1 ; 348: lw $t161, val8 # $t161 is 2147483647
[00400294] lui $t162, 0x1 ; 349: lw $t162, val8 # $t162 is 2147483647
[00400298] lui $t163, 0x1 ; 350: lw $t163, val8 # $t163 is 2147483647
[0040029c] lui $t164, 0x1 ; 351: lw $t164, val8 # $t164 is 2147483647
[004002a0] lui $t165, 0x1 ; 352: lw $t165, val8 # $t165 is 2147483647
[004002a4] lui $t166, 0x1 ; 353: lw $t166, val8 # $t166 is 2147483647
[004002a8] lui $t167, 0x1 ; 354: lw $t167, val8 # $t167 is 2147483647
[004002ac] lui $t168, 0x1 ; 355: lw $t168, val8 # $t168 is 2147483647
[004002b0] lui $t169, 0x1 ; 356: lw $t169, val8 # $t169 is 2147483647
[004002b4] lui $t170, 0x1 ; 357: lw $t170, val8 # $t170 is 2147483647
[004002b8] lui $t171, 0x1 ; 358: lw $t171, val8 # $t171 is 2147483647
[004002bc] lui $t172, 0x1 ; 359: lw $t172, val8 # $t172 is 2147483647
[004002c0] lui $t173, 0x1 ; 360: lw $t173, val8 # $t173 is 2147483647
[004002c4] lui $t174, 0x1 ; 361: lw $t174, val8 # $t174 is 2147483647
[004002c8] lui $t175, 0x1 ; 362: lw $t175, val8 # $t175 is 2147483647
[004002cc] lui $t176, 0x1 ; 363: lw $t176, val8 # $t176 is 2147483647
[004002d0] lui $t177, 0x1 ; 364: lw $t177, val8 # $t177 is 2147483647
[004002d4] lui $t178, 0x1 ; 365: lw $t178, val8 # $t178 is 2147483647
[004002d8] lui $t179, 0x1 ; 366: lw $t179, val8 # $t179 is 2147483647
[004002dc] lui $t180, 0x1 ; 367: lw $t180, val8 # $t180 is 2147483647
[004002e0] lui $t181, 0x1 ; 368: lw $t181, val8 # $t181 is 2147483647
[004002e4] lui $t182, 0x1 ; 369: lw $t182, val8 # $t182 is 2147483647
[004002e8] lui $t18
```

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, $2^{15} \Rightarrow 32768 - 1 = 32767(00007fff)$
 $-2^{15} = -32768(ffff8000)$

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.))