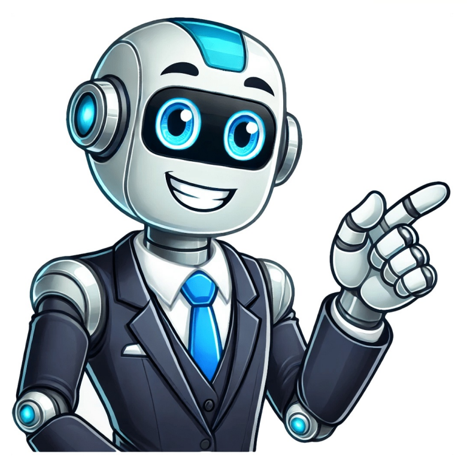
Click to verify



How cmp works in assembly

Yes, that's correct, except for *x in your C code and missing assembly code. In C, you define variable types (signed/unsigned) upon declaration, such as int x or unsigned int x, whereas in assembly, the distinction between signed and unsigned variables is made through different conditional jumps: For signed variables, jg/jl/jge/jle; for unsigned variables, ja/jb/jae/jbe. I'm trying to understand assembly to solve a puzzle, but I encountered instructions: cmpl $0x7, 0x14(%rsp); ja 0x401230. I think it's comparing the value of 0x14(%rsp) (-7380), which is unsigned, and if it's greater than 7, then jump, which might not be what you want since (unsigned)-7380 > 7 would indeed jump. It seems you're flipping arguments or jumping with an incorrect condition. Regarding your assembly program, I'll focus on the comparison. The cmp command compares two operands: AX and BX. Since it's signed division, ax/bx is stored in dx, then decremented to zero. In this context, the comparison works by performing a subtraction of bx from ax, setting flags (OF, SF, ZF, AF, PF, CF) according to the result. In general, your program prints a number. The comparison instructions in x86 assembly language, such as CMP and CMN, perform operations on two operands but do not store the result directly. Instead, they update flags that can be used for conditional branching. The CMP instruction calculates DESTINATION - SOURCE, setting flags based on this subtraction. For instance, if comparing a register to zero with CMP eax, 0, the result is equal if the register holds a non-zero value and false otherwise. CMN is similar but allows comparison against small negative values. The TEQ instruction is analogous to TST but uses an EOR operation instead of AND, making it useful for checking bit equivalences without affecting the Carry flag. TST itself performs a bitwise AND between two operands, updating flags based on this result, and can be used to test specific bits in data. To access the results of these operations, one typically relies on the condition flags set by CMP or TEQ/TST instructions. In Intel syntax, CMP is often represented as cmp dest, src, while AT&T syntax uses cmp $src, %dest. Understanding the behavior of these instructions and their implications for conditional jumps in x86 assembly is crucial for effective programming in this environment. CMP instructions work similarly to SUB instructions but without changing operands, instead subtracting source operand from destination operand and updating flags only. For example: CMP AX, BX ; Performs (AX-BX) and updates flags, leaving source and destination unchanged. Unlike jumps alone, the comparison is meaningless as there's no decision-making involved if you're not taking an action based on it. Jumps are used with conditional jump instructions to take decisions. There are two main types of jump instructions: unconditional and conditional. Unconditional jumps force the processor to start executing instructions from a specific address unconditionally, as shown in this code example: JMP 03h. This assigns the address 0003 to the instruction pointer. Conditional jumps, on the other hand, require a condition to be met before starting execution at a specified address. For instance: JC L1 ; Jump to label L1 if CF (carry flag) is set. The difference between signed and unsigned number comparisons can also be explained using conditional jump instructions. The `CMP` (compare) instruction is used to compare two operands and sets status flags in the EFLAGS register according to the result. There are different forms of this instruction depending on the mode and registers involved. In signed mode, the comparison is performed as subtraction, while in unsigned mode, it's done using bitwise comparison. The `CMP` instruction takes an immediate value as a second operand and sign-extends it to the length of the first operand. The result is used to set status flags in the EFLAGS register, which are then used by other instructions such as `Jcc`, `CMOVcc`, and `SETcc`. In 64-bit mode, the default operation size is 32 bits, but using the REX.R prefix allows access to additional registers (R8-R15), while the REX.W prefix promotes the operation to 64 bits. The `CMP` instruction sets several flags: CF (carry flag), OF (overflow flag), SF (sign flag), ZF (zero flag), AF (alternate flag), and PF (parity flag). Note that some registers, such as AH, BH, CH, DH, cannot be accessed in 64-bit mode with a REX prefix. Protected Mode Exceptions If an address is outside the segment limits (CS, DS, ES, FS, or GS) or a NULL segment selector is detected. #SS(0) If an address is outside the SS segment limit. #PF(fault-code) If a page fault occurs. #AC(0) If alignment checking is enabled and an unaligned reference is made while the current privilege level is 3. #UD If the LOCK prefix is used. Real-Address Mode Exceptions #GP If an address is outside the CS, DS, ES, FS, or GS segment limit. #SS If an address is outside the SS segment limit. Virtual-8086 Mode Exceptions #GP(0) If an address is outside the CS, DS, ES, FS, or GS segment limit. #SS(0) If an address is outside the SS segment limit. #PF(fault-code) If a page fault occurs. #AC(0) If alignment checking is enabled and an unaligned reference is made. #UD If the LOCK prefix is used. Compatibility Mode Exceptions Same exceptions as in protected mode. 64-Bit Mode Exceptions #SS(0) If a memory address referencing the SS segment is in a non-canonical form. #GP(0) If the memory address is in a non-canonical form. #PF(fault-code) If a page fault occurs. #AC(0) If alignment checking is enabled and an unaligned reference is made while the current privilege level is 3. #UD If the LOCK prefix is used. Conditional Execution Conditional execution in assembly language involves several looping and branching instructions, which change the flow of control in a program. Two scenarios for conditional execution are: 1. Unconditional jump (JMP instruction) 2. Conditional jump (set of jump instructions j depending upon the condition) The CMP instruction compares two operands and is used along with conditional jump instructions for decision making. Whether the counter value has reached the specified limit for loop iterations. To achieve this, we can use a condition check. For instance: INC EDX CMP EDX, 10 ; Check if the counter is at or below 10 JLE LP1 ; If it's less than or equal to 10, jump back to LP1 As mentioned earlier, conditional execution involves transferring control flow using instructions like JMP. This can be done forward to execute new instructions or backward to re-execute previous steps. The JMP instruction provides a label name where the control flow is transferred immediately. Its syntax is: JMP label Here's an example illustrating the use of JMP: MOV AX, 00 ; Initialize AX to 0 MOV BX, 00 ; Initialize BX to 0 MOV CX, 01 ; Initialize CX to 1 L20: ADD AX, 01 ; Increment AX ADD BX, AX ; Add AX to BX SHL CX, 1 ; Shift left CX, doubling its value JMP L20 ; Repeat the statements Conditional Jump Instructions: If a specified condition is met in conditional jump, control flow is transferred to a target instruction. The following are examples of conditional jump instructions used on signed data for arithmetic operations: - JE/JZ: Jump Equal or Jump Zero (ZF) - JNE/JNZ: Jump Not Equal or Jump Not Zero (ZF) - JG/JNLE: Jump Greater or Jump Not Less/Equal (OF, SF, ZF) - JGE/JNL: Jump Greater/Equal or Jump Not Less (OF, SF) - JL/JNGE: Jump Less or Jump Not Greater/Equal (OF, SF) For unsigned data used for logical operations: - JE/JZ: Jump Equal or Jump Zero (ZF) - JNE/JNZ: Jump not Equal or Jump Not Zero (ZF) - JA/JNBE: Jump Above or Jump Not Below/Equal (CF, ZF) - JAE/JNB: Jump Above/Equal or Jump Not Below (CF) And special uses that check the value of flags: - JC: Jump If Carry (CF) - JNC: Jump If No Carry (CF) - JO: Jump If Overflow (OF) - JNO: Jump If No Overflow (OF) - JP/JPE: Jump Parity or Jump Parity Even (PF) - JNP/JPO: Jump No Parity or Jump Parity Odd (PF) - JS: Jump Sign (negative value) (SF) - JNS: Jump No Sign (positive value) (SF) The syntax for the J set of instructions: Example: CMP AL, BL JE EQUAL ... EQUAL: ... A program that displays the largest of three variables using conditional jump instructions is provided below. The variables are double-digit numbers with values 47, 22, and 31. section .text global _start _start: mov ecx, [num1] cmp ecx, [num2] jg check_third_num mov ecx, [num2] check_third_num: cmp ecx, [num3] jg _exit mov ecx, [num3] _exit: mov [largest], ecx mov ecx,msg mov edx, len mov ebx,1 ;file descriptor (stdout) mov eax,4 ;system call number (sys_write) int 0x80 ;call kernel mov ecx,largest mov edx, 2 mov ebx,1 ;file descriptor (stdout) mov eax,4 ;system call number The cmp instruction in ARM assembly compares two operands by subtracting the value of Operand2 from the value in Rn, discarding the result. It serves the same purpose as a SUB instruction but without modifying operands, influencing only the Zero Flag (ZF) and Carry Flag (CF).