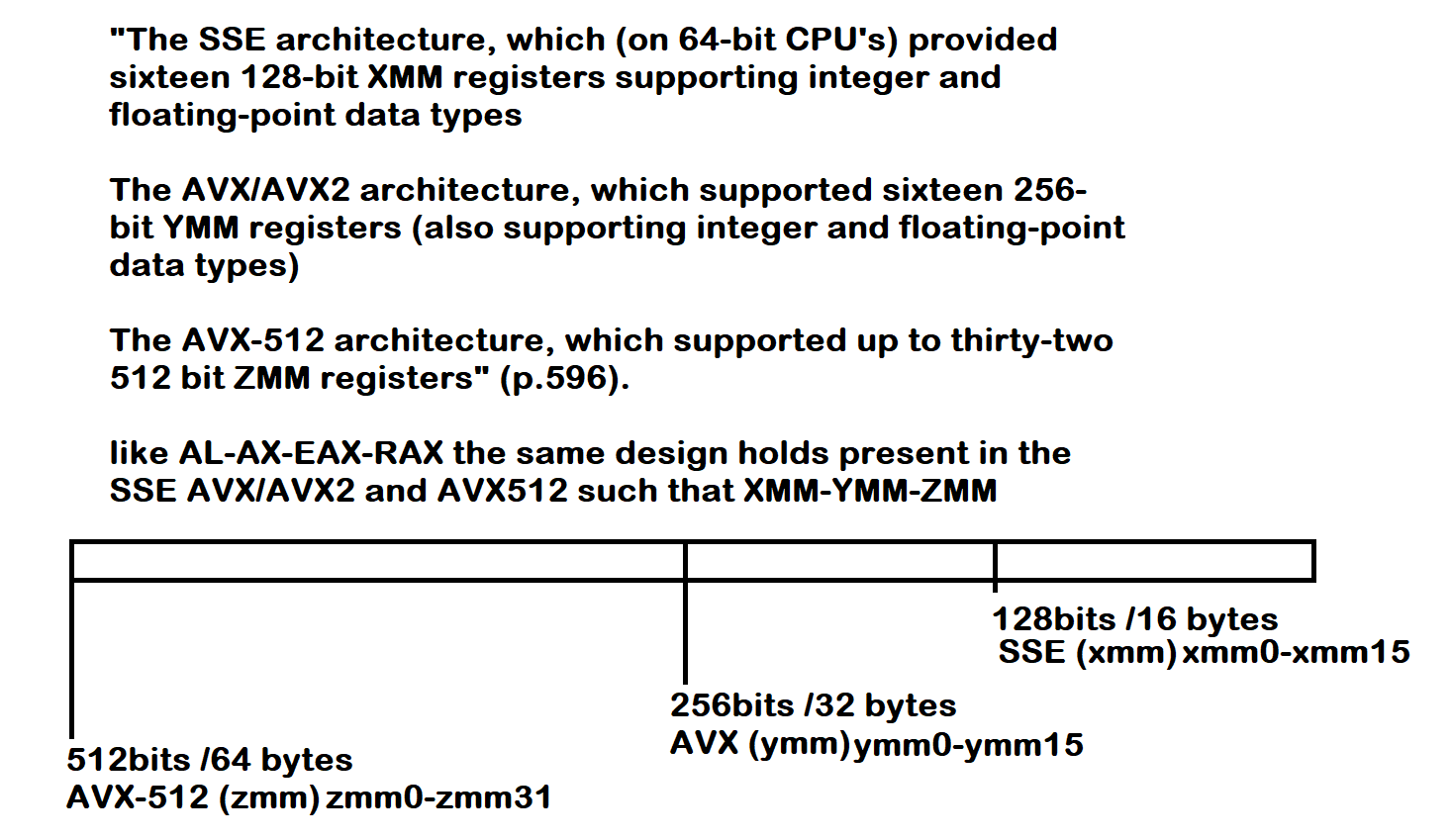
Jonathan Lee

Professor: Devin Cook

Class: CSC35

Date: 5-21-23

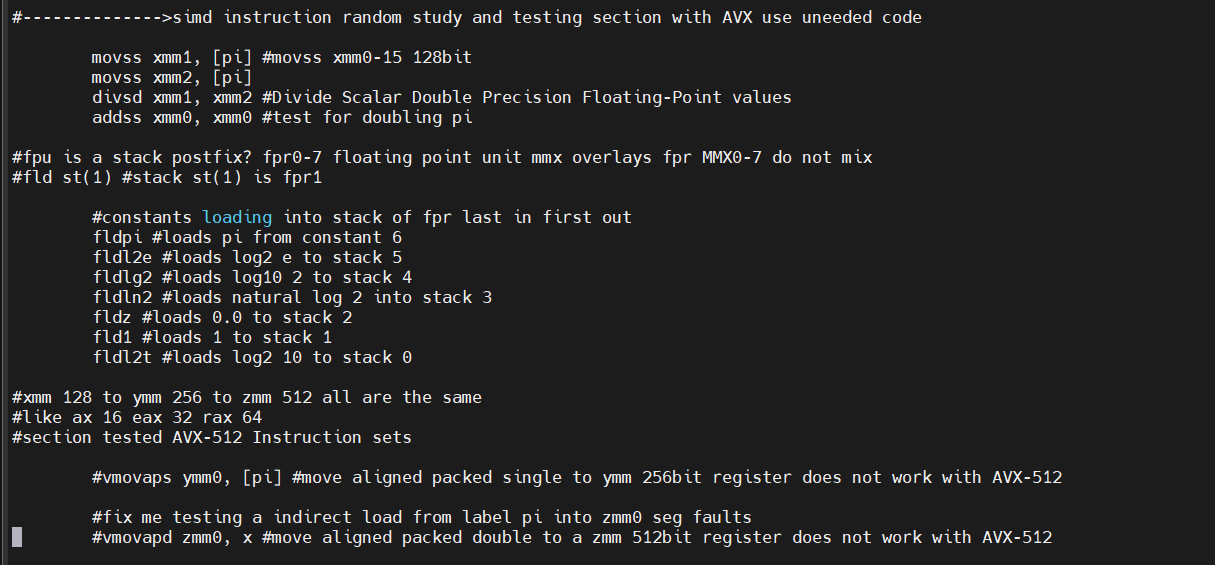
Quick Overview of CPUID and 128-512 bit registers



Hyde P.596

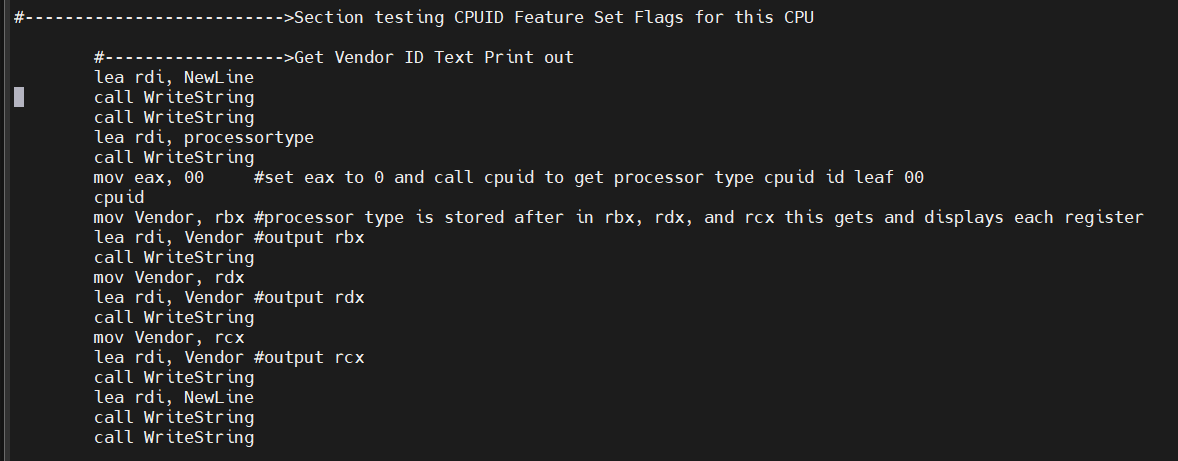
Does the Herky processor support AVX-512. . . This question caused me some confusion as I tested many commands to learn about higher level bit registers.





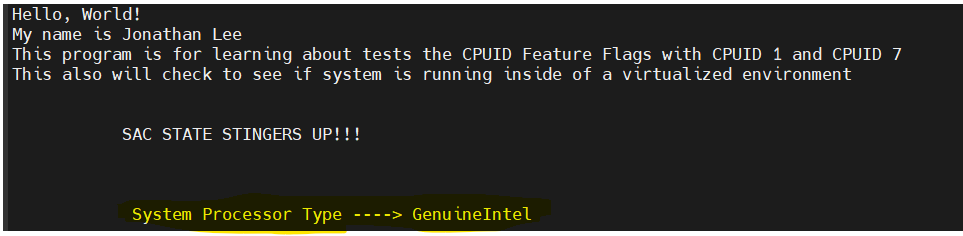
In the images about are just some of the commands and code I tested to see if I could get any AVX-512 mnemonics to work. In the end I was able to work with the xmm 128-bit registers. I thought there has to be an easier way to check instruction sets, without mindless tests of each and every command. What I really mean is that while I was reading about something called 512-bit registers, I wanted to attempt to code something that worked with this hypothetical Rachmaninoff register the “ZMM”. My wife said just practice the basics, again after many hours, I learned she was right.

While in my hypothetical rabbit hole of code, I learned after all the tests, we have to use CPUID to find the answer about AVX-512 instruction set support. What CPUID does is tell us what instruction sets are supported by the processor, it can display the vendor id, it can even tell you if you are working in a virtual environment.



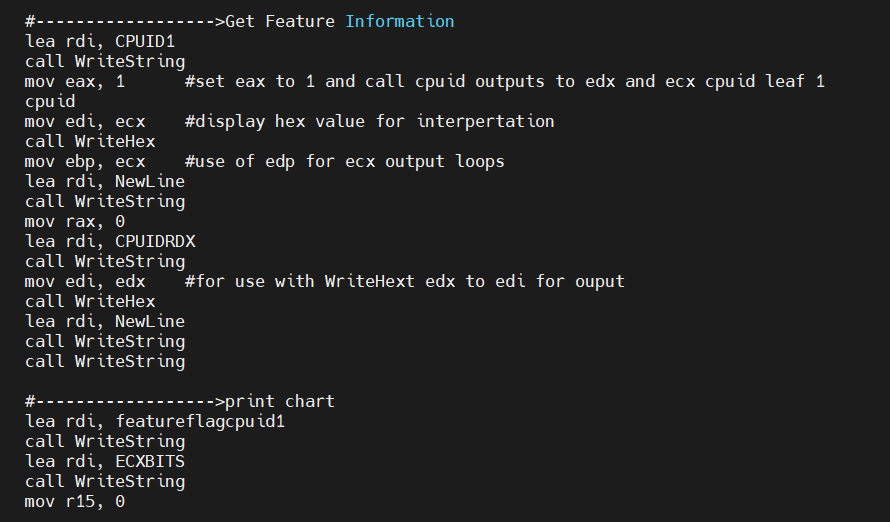
This code sets EAX to 00 for use what CPUID leaf 00. CPUID uses what are called leafs when the user calls CPUID.

CPUID with leaf 00 will return the vendor ID. It will store the id into RBX, RDX, and RCX. CPUID mnemonic is kind of like a system call, were we load something into a register (EAX for CPUID), whereas (RAX for SYSCALL) and run the command and it returns its reply into other registers.



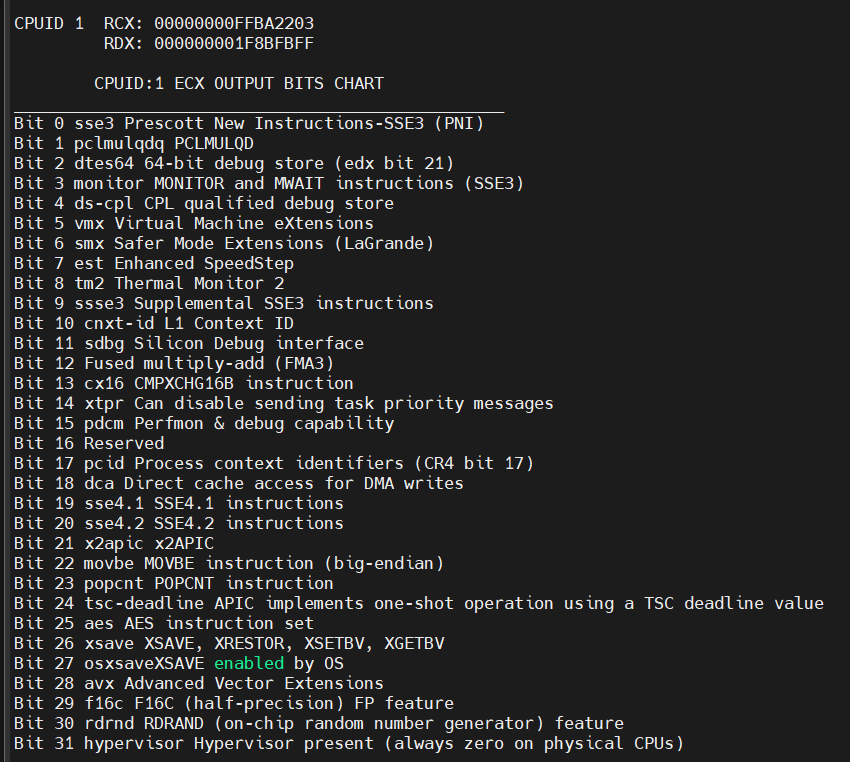
As we can see we have a Genuine Intel processor.

Next, we will set the CPUID leaf to 1. The results will be stored into RCX and RDX in hex decimal. Each bit that is turned on (showing 1) gives us information as to what is supported by this Herky processor.

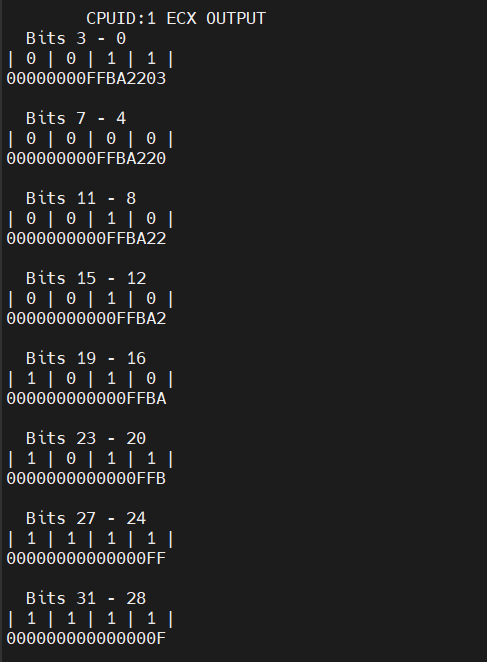


This code will get the feature information. With the help of a chart, we can check what is supported.

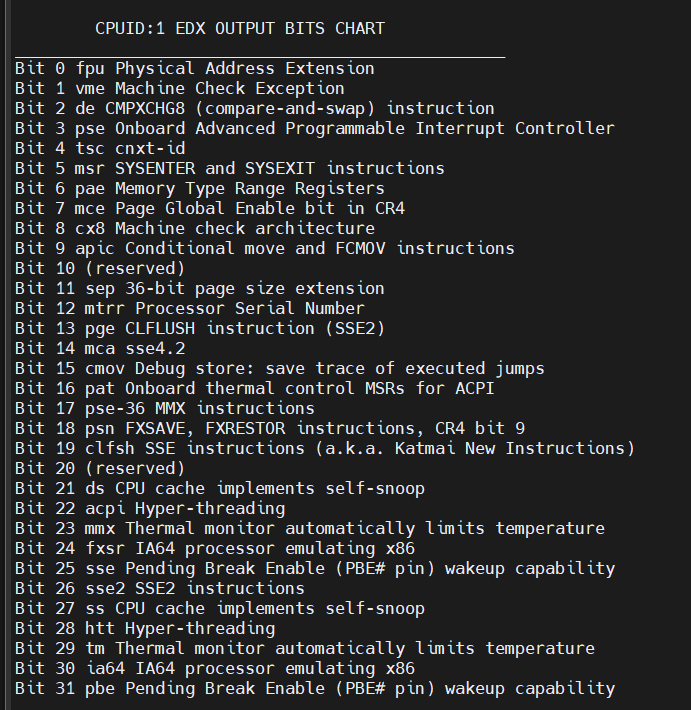
#REF: charts https://en.wikipedia.org/wiki/CPUID



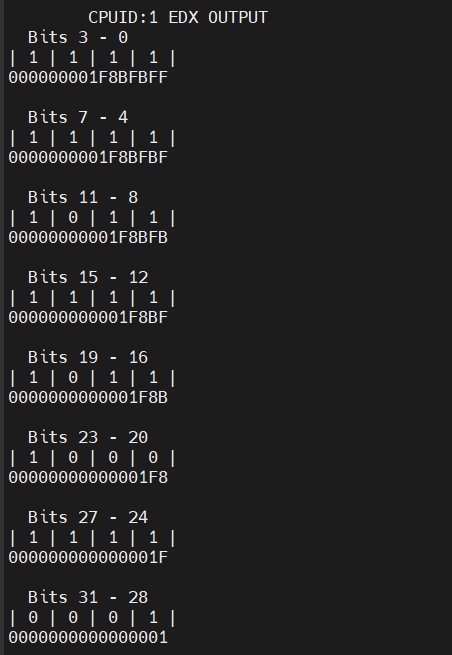
As we can see at the top of this image registers RCX, and RDX have been populated. The chart will act as a reference to what flags are turned on.



This output is the broken-down hex value. It shows each bit to use with the chart above. Each line is a different hex value. Example 0x3 hex = 0011 binary, with the chart as reference flags one and two enabled mean that this Herky processor supports sse3 prescott new instructions-sse3 and pclmulqdq.

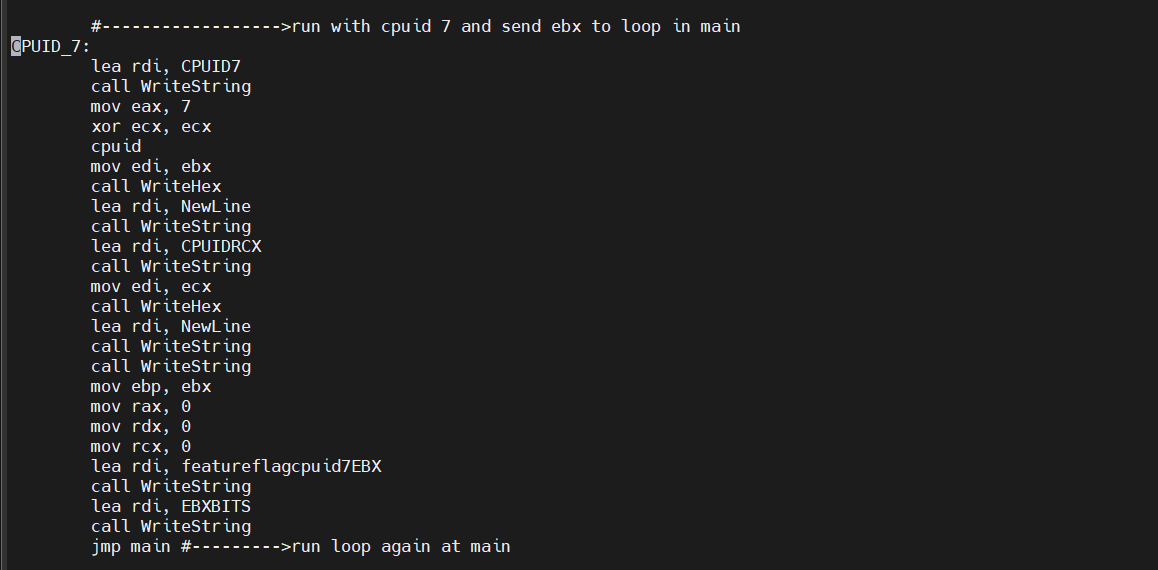


Let us now turn to the EDX bits

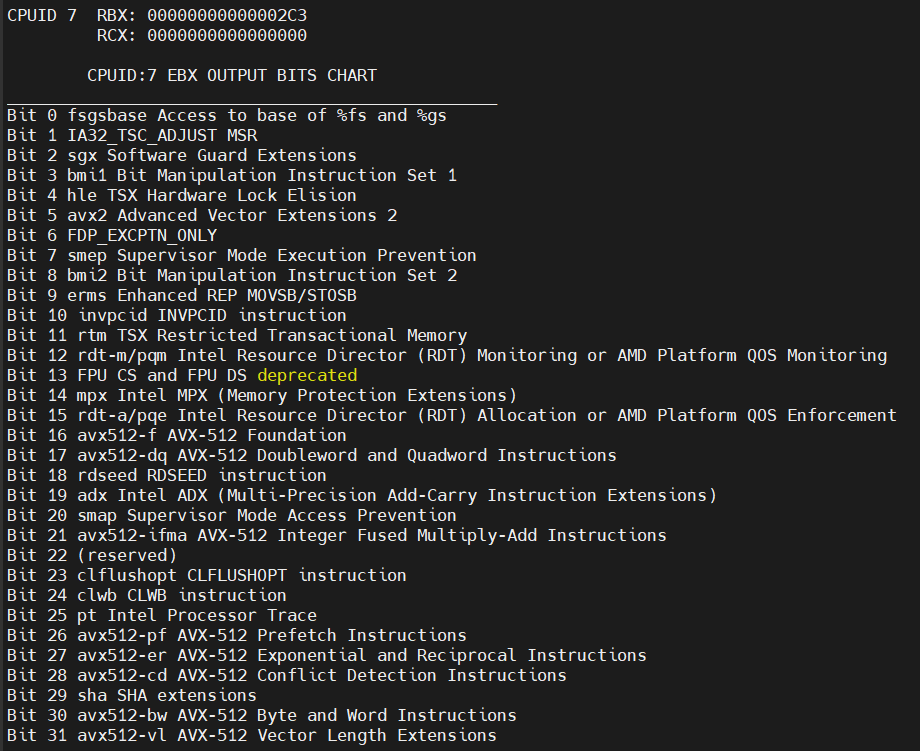


This ends the CPUID leaf 1.

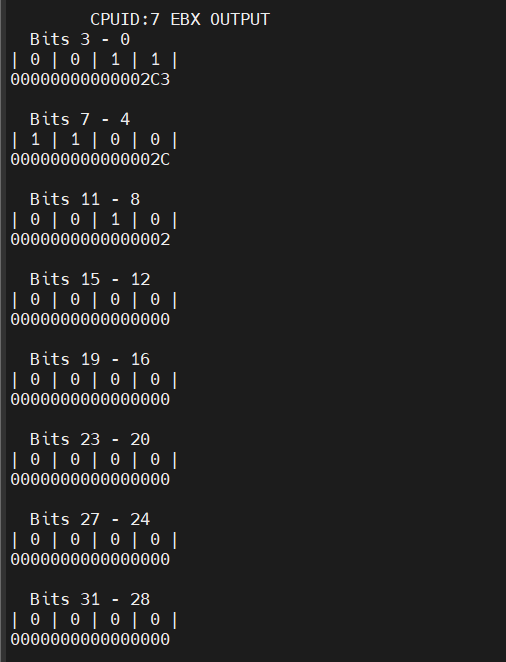
Now let’s do CPUID leaf 7 to get the extended feature flags.



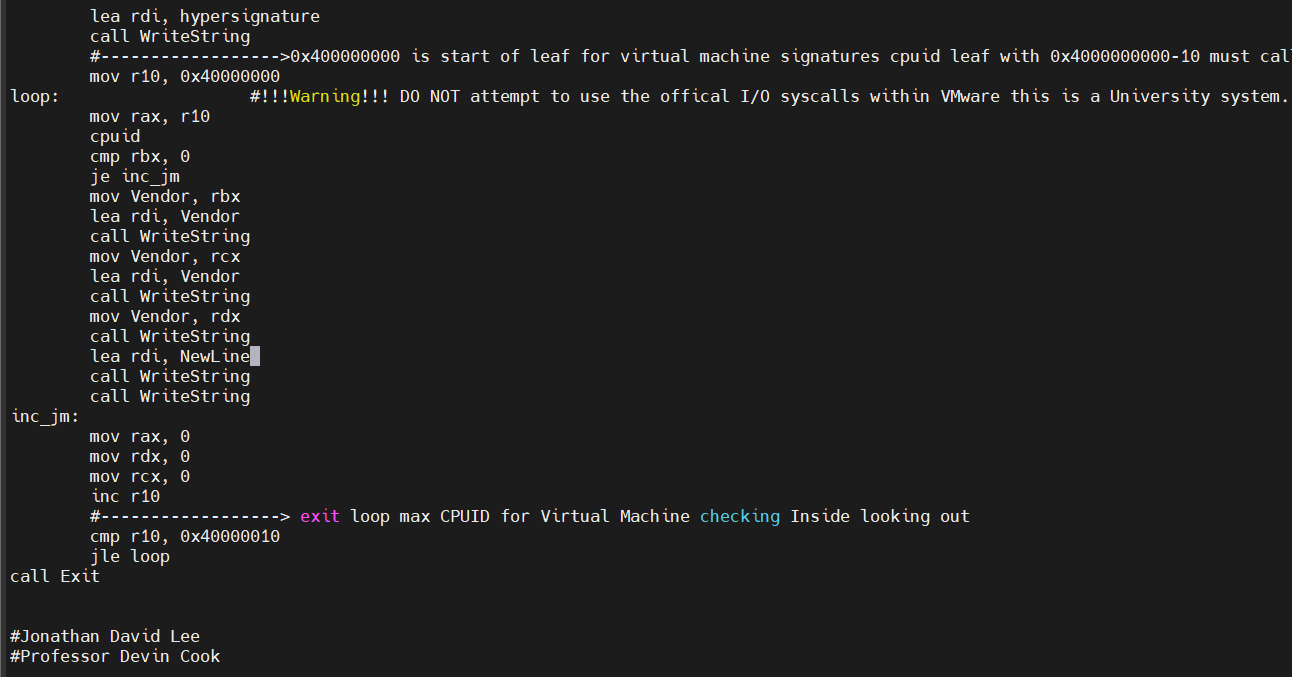
Here is our code. Notice we loaded EAX with 7 and XOR ECX, ECX and called CPUID again. Again, we loop back for our bits chart after we have our hex value.



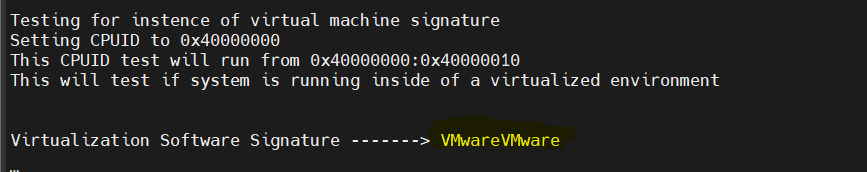
We now have our hex values populated into RBX and RCX for this leaf. RBX shows 0x2c3 at the top of the image.



This is our bit break down for this program. Bits 26-31 show no AVX-512 support. Therefore, the Herky processor does not have AVX-512 instruction sets.



We can even use sets of CPUID leafs like in this image. This set of leafs will look at every CPUID from a specific range to have the CPU tell us what Virtual machine software signature it has. This also stores the results into RBX, RCX, and RDX.



The Herky processor is running inside VMware, meaning we are inside a virtualized environment.

**WORKS CITED:**

“CPUID.” *Wikipedia*, 11 May 2023, en.wikipedia.org/wiki/CPUID.

Hyde, Randall. “11. SMID INSTRUCTIONS.” *The Art of 64-Bit Assembly*, No Starch Press Inc, San Francisco, 2022, pp. 595–605.

CODE ATTACHED:

#Jonathan Lee

#Professor Devin Cook

#CSC35 SEC 1

#CPUID.asm

#This practice program will use CPUID to print processor capabilities as well as check to see if you are inside an instance of a virtural machine

#4-5-2023

.intel\_syntax noprefix

.data

Vendor:

.quad 00000000

Bits:

.ascii " Bits \0"

Bar:

.ascii "| \0"

Dash:

.ascii " - \0"

processortype:

.ascii " System Processor Type ----> \0"

featureflagcpuid1: #REF: charts https://en.wikipedia.org/wiki/CPUID

.ascii " CPUID:1 ECX OUTPUT BITS CHART\n"

.ascii "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n"

.ascii "Bit 0 sse3 Prescott New Instructions-SSE3 (PNI)\n"

.ascii "Bit 1 pclmulqdq PCLMULQD\n"

.ascii "Bit 2 dtes64 64-bit debug store (edx bit 21)\n"

.ascii "Bit 3 monitor MONITOR and MWAIT instructions (SSE3)\n"

.ascii "Bit 4 ds-cpl CPL qualified debug store\n"

.ascii "Bit 5 vmx Virtual Machine eXtensions\n"

.ascii "Bit 6 smx Safer Mode Extensions (LaGrande)\n"

.ascii "Bit 7 est Enhanced SpeedStep\n"

.ascii "Bit 8 tm2 Thermal Monitor 2\n"

.ascii "Bit 9 ssse3 Supplemental SSE3 instructions\n"

.ascii "Bit 10 cnxt-id L1 Context ID\n"

.ascii "Bit 11 sdbg Silicon Debug interface\n"

.ascii "Bit 12 Fused multiply-add (FMA3)\n"

.ascii "Bit 13 cx16 CMPXCHG16B instruction\n"

.ascii "Bit 14 xtpr Can disable sending task priority messages\n"

.ascii "Bit 15 pdcm Perfmon & debug capability\n"

.ascii "Bit 16 Reserved\n"

.ascii "Bit 17 pcid Process context identifiers (CR4 bit 17)\n"

.ascii "Bit 18 dca Direct cache access for DMA writes\n"

.ascii "Bit 19 sse4.1 SSE4.1 instructions\n"

.ascii "Bit 20 sse4.2 SSE4.2 instructions\n"

.ascii "Bit 21 x2apic x2APIC\n"

.ascii "Bit 22 movbe MOVBE instruction (big-endian)\n"

.ascii "Bit 23 popcnt POPCNT instruction\n"

.ascii "Bit 24 tsc-deadline APIC implements one-shot operation using a TSC deadline value\n"

.ascii "Bit 25 aes AES instruction set\n"

.ascii "Bit 26 xsave XSAVE, XRESTOR, XSETBV, XGETBV\n"

.ascii "Bit 27 osxsaveXSAVE enabled by OS\n"

.ascii "Bit 28 avx Advanced Vector Extensions\n"

.ascii "Bit 29 f16c F16C (half-precision) FP feature\n"

.ascii "Bit 30 rdrnd RDRAND (on-chip random number generator) feature\n"

.ascii "Bit 31 hypervisor Hypervisor present (always zero on physical CPUs)\n\n\0"

featureflagcpuid1EDX:

.ascii " CPUID:1 EDX OUTPUT BITS CHART\n"

.ascii "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n"

.ascii "Bit 0 fpu Physical Address Extension\n"

.ascii "Bit 1 vme Machine Check Exception\n"

.ascii "Bit 2 de CMPXCHG8 (compare-and-swap) instruction\n"

.ascii "Bit 3 pse Onboard Advanced Programmable Interrupt Controller\n"

.ascii "Bit 4 tsc cnxt-id\n"

.ascii "Bit 5 msr SYSENTER and SYSEXIT instructions\n"

.ascii "Bit 6 pae Memory Type Range Registers\n"

.ascii "Bit 7 mce Page Global Enable bit in CR4\n"

.ascii "Bit 8 cx8 Machine check architecture\n"

.ascii "Bit 9 apic Conditional move and FCMOV instructions\n"

.ascii "Bit 10 (reserved)\n"

.ascii "Bit 11 sep 36-bit page size extension\n"

.ascii "Bit 12 mtrr Processor Serial Number\n"

.ascii "Bit 13 pge CLFLUSH instruction (SSE2)\n"

.ascii "Bit 14 mca sse4.2\n"

.ascii "Bit 15 cmov Debug store: save trace of executed jumps\n"

.ascii "Bit 16 pat Onboard thermal control MSRs for ACPI\n"

.ascii "Bit 17 pse-36 MMX instructions\n"

.ascii "Bit 18 psn FXSAVE, FXRESTOR instructions, CR4 bit 9\n"

.ascii "Bit 19 clfsh SSE instructions (a.k.a. Katmai New Instructions)\n"

.ascii "Bit 20 (reserved)\n"

.ascii "Bit 21 ds CPU cache implements self-snoop\n"

.ascii "Bit 22 acpi Hyper-threading\n"

.ascii "Bit 23 mmx Thermal monitor automatically limits temperature\n"

.ascii "Bit 24 fxsr IA64 processor emulating x86\n"

.ascii "Bit 25 sse Pending Break Enable (PBE# pin) wakeup capability\n"

.ascii "Bit 26 sse2 SSE2 instructions\n"

.ascii "Bit 27 ss CPU cache implements self-snoop\n"

.ascii "Bit 28 htt Hyper-threading\n"

.ascii "Bit 29 tm Thermal monitor automatically limits temperature\n"

.ascii "Bit 30 ia64 IA64 processor emulating x86\n"

.ascii "Bit 31 pbe Pending Break Enable (PBE# pin) wakeup capability\n\n\0"

featureflagcpuid7EBX:

.ascii " CPUID:7 EBX OUTPUT BITS CHART\n"

.ascii "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n"

.ascii "Bit 0 fsgsbase Access to base of %fs and %gs\n"

.ascii "Bit 1 IA32\_TSC\_ADJUST MSR\n"

.ascii "Bit 2 sgx Software Guard Extensions\n"

.ascii "Bit 3 bmi1 Bit Manipulation Instruction Set 1\n"

.ascii "Bit 4 hle TSX Hardware Lock Elision\n"

.ascii "Bit 5 avx2 Advanced Vector Extensions 2\n"

.ascii "Bit 6 FDP\_EXCPTN\_ONLY\n"

.ascii "Bit 7 smep Supervisor Mode Execution Prevention\n"

.ascii "Bit 8 bmi2 Bit Manipulation Instruction Set 2\n"

.ascii "Bit 9 erms Enhanced REP MOVSB/STOSB\n"

.ascii "Bit 10 invpcid INVPCID instruction\n"

.ascii "Bit 11 rtm TSX Restricted Transactional Memory\n"

.ascii "Bit 12 rdt-m/pqm Intel Resource Director (RDT) Monitoring or AMD Platform QOS Monitoring\n"

.ascii "Bit 13 FPU CS and FPU DS deprecated\n"

.ascii "Bit 14 mpx Intel MPX (Memory Protection Extensions)\n"

.ascii "Bit 15 rdt-a/pqe Intel Resource Director (RDT) Allocation or AMD Platform QOS Enforcement\n"

.ascii "Bit 16 avx512-f AVX-512 Foundation\n"

.ascii "Bit 17 avx512-dq AVX-512 Doubleword and Quadword Instructions\n"

.ascii "Bit 18 rdseed RDSEED instruction\n"

.ascii "Bit 19 adx Intel ADX (Multi-Precision Add-Carry Instruction Extensions)\n"

.ascii "Bit 20 smap Supervisor Mode Access Prevention\n"

.ascii "Bit 21 avx512-ifma AVX-512 Integer Fused Multiply-Add Instructions\n"

.ascii "Bit 22 (reserved)\n"

.ascii "Bit 23 clflushopt CLFLUSHOPT instruction\n"

.ascii "Bit 24 clwb CLWB instruction\n"

.ascii "Bit 25 pt Intel Processor Trace\n"

.ascii "Bit 26 avx512-pf AVX-512 Prefetch Instructions\n"

.ascii "Bit 27 avx512-er AVX-512 Exponential and Reciprocal Instructions\n"

.ascii "Bit 28 avx512-cd AVX-512 Conflict Detection Instructions\n"

.ascii "Bit 29 sha SHA extensions\n"

.ascii "Bit 30 avx512-bw AVX-512 Byte and Word Instructions\n"

.ascii "Bit 31 avx512-vl AVX-512 Vector Length Extensions\n\n\0"

CPUID1:

.ascii "CPUID 1 RCX: \0"

CPUIDRDX:

.ascii " RDX: \0"

CPUID7:

.ascii "CPUID 7 RBX: \0"

CPUIDRCX:

.ascii " RCX: \0"

ECXBITS:

.ascii " CPUID:1 ECX OUTPUT\n\0"

EDXBITS:

.ascii " CPUID:1 EDX OUTPUT\n\0"

EBXBITS:

.ascii " CPUID:7 EBX OUTPUT\n\0"

NewLine:

.ascii "\n\0"

Zero:

.ascii "0 | \0"

One:

.ascii "1 | \0"

Name:

.ascii "Hello, World!\nMy name is Jonathan Lee\nThis program is for learning about tests the CPUID Feature Flags with CPUID 1 and CPUID 7\n"

.ascii "This also will check to see if system is running inside of a virtualized environment\n"

.ascii "\n\n SAC STATE STINGERS UP!!!\n\n\0"

hypersignature:

.ascii "Testing for instence of virtual machine signature\n"

.ascii "Setting CPUID to 0x40000000\n"

.ascii "This CPUID test will run from 0x40000000:0x40000010\n"

.ascii "This will test if system is running inside of a virtualized environment\n\n\n"

.ascii "Virtualization Software Signature -------> \0"

backarrow:

.ascii " <-------\0"

.text

.global \_start

\_start:

call ClearScreen

lea rdi, Name

call WriteString

#-------------------------------------------->------MAIN PROGRAM AREA------------------------->--->--->

#-------------------------->Section testing CPUID Feature Set Flags for this CPU

#------------------>Get Vendor ID Text Print out

lea rdi, NewLine

call WriteString

call WriteString

lea rdi, processortype

call WriteString

mov eax, 00 #set eax to 0 and call cpuid to get processor type cpuid id leaf 00

cpuid

mov Vendor, rbx #processor type is stored after in rbx, rdx, and rcx this gets and displays each register

lea rdi, Vendor #output rbx

call WriteString

mov Vendor, rdx

lea rdi, Vendor #output rdx

call WriteString

mov Vendor, rcx

lea rdi, Vendor #output rcx

call WriteString

lea rdi, NewLine

call WriteString

call WriteString

#------------------>Get Feature Information

lea rdi, CPUID1

call WriteString

mov eax, 1 #set eax to 1 and call cpuid outputs to edx and ecx cpuid leaf 1

cpuid

mov edi, ecx #display hex value for interpertation

call WriteHex

mov ebp, ecx #use of edp for ecx output loops

lea rdi, NewLine

call WriteString

mov rax, 0

lea rdi, CPUIDRDX

call WriteString

mov edi, edx #for use with WriteHext edx to edi for ouput

call WriteHex

lea rdi, NewLine

call WriteString

call WriteString

#------------------>print chart

lea rdi, featureflagcpuid1

call WriteString

lea rdi, ECXBITS

call WriteString

mov r15, 0

#------------------>Main Loop for use with nibbles to binary

main:

mov r13, 0

mov r14, 0

mov r9, 0

lea rdi, Bits

call WriteString

mov r13, 3

mov rdi, r13

call WriteInt

lea rdi, Dash

call WriteString

mov rdi, r14

call WriteInt

lea rdi, NewLine

call WriteString

lea rdi, Bar

call WriteString

# Possible fix me this does not use stacks will update when stacks and queues are leared with assembly

mov edx, 8

mov ecx, 4

mov r9b, bpl

mov rax, 0

mov al, bpl

shl al, 4

shr al, 4

mov r9b, al

loop\_Binary\_First\_Nibble: #issue with lsb had to manually do this loop for that first nibble shifts right and left because I am doing sets of nibbles for visibility

cmp r9b, dl

jl printZero

printOne:

sub r9b, dl

lea rdi, One

call WriteString

jmp end\_

printZero:

lea rdi, Zero

call WriteString

end\_:

mov al, dl

mov bl, 2

div bl

mov dl,al

loop loop\_Binary\_First\_Nibble

#------------------>prints out hex value of ebp

lea rdi, NewLine

call WriteString

mov edi, ebp

call WriteHex

lea rdi, NewLine

call WriteString

call WriteString

mov r12, 7 #------->loopshift counter loop 7 times

#loop after first nibble removed this will cut the nibble and continue

loopshift:

#---------->Prints Bits range

lea rdi, Bits

call WriteString

add r13, 4

mov rdi, r13

call WriteInt

lea rdi, Dash

call WriteString

add r14, 4

mov rdi, r14

call WriteInt

lea rdi, NewLine

call WriteString

lea rdi, Bar

call WriteString

#---------->set up loop again

mov edx, 8

mov ecx, 4

mov r9b, bpl

mov rax, r9

shr eax, 4

mov r9b, al

mov rax, 0

#------------------>loop for nibble

loop\_Binary:

cmp r9b, dl

jl print\_Zero

print\_One:

sub r9b, dl

lea rdi, One

call WriteString

jmp end

print\_Zero:

lea rdi, Zero

call WriteString

end:

mov al, dl

mov bl, 2

div bl

mov dl,al

loop loop\_Binary

#------------------>remove 1 nibble

shr ebp, 4

lea rdi, NewLine

call WriteString

mov edi, ebp

call WriteHex

lea rdi, NewLine

call WriteString

call WriteString

inc r15

cmp r15, 14 #------>mov to cpuid 7 pass run again area

je CPUID\_7

cmp r15, 20 #------>exit condition

jg Over

dec r12 #---------->decrement r12 1

cmp r12, 0 #------->check has it reached 0

jg loopshift #----->jump if greater than 0 iterate loopshift again

#------------------>run again with edx once loop reaches 0

mov eax, 1

cpuid

mov ebp, edx

mov rax, 0

mov rdx, 0

mov rcx, 0

lea rdi, featureflagcpuid1EDX

call WriteString

lea rdi, EDXBITS

call WriteString

#------------------>reset loop to main for edx cpuid 1 run

jmp main #--------->jumping to main

#------------------>run with cpuid 7 and send ebx to loop in main

CPUID\_7:

lea rdi, CPUID7

call WriteString

mov eax, 7

xor ecx, ecx

cpuid

mov edi, ebx

call WriteHex

lea rdi, NewLine

call WriteString

lea rdi, CPUIDRCX

call WriteString

mov edi, ecx

call WriteHex

lea rdi, NewLine

call WriteString

call WriteString

mov ebp, ebx

mov rax, 0

mov rdx, 0

mov rcx, 0

lea rdi, featureflagcpuid7EBX

call WriteString

lea rdi, EBXBITS

call WriteString

jmp main #--------->run loop again at main

#------------------>Loop checks for Virtual Machine Environment software signature with CPUID"

Over: #------------------>Exit jump point

lea rdi, hypersignature

call WriteString

#------------------>0x400000000 is start of leaf for virtual machine signatures cpuid leave with 0x4000000000-10 must call each for every vendor type

mov r10, 0x40000000

loop: #!!!Warning!!! DO NOT attempt to use the offical I/O syscalls within VMware this is a University system. Just leave it alone. Just get signature value.

mov rax, r10

cpuid

cmp rbx, 0

je inc\_jm

mov Vendor, rbx

lea rdi, Vendor

call WriteString

mov Vendor, rcx

lea rdi, Vendor

call WriteString

mov Vendor, rdx

call WriteString

lea rdi, NewLine

call WriteString

call WriteString

inc\_jm:

mov rax, 0

mov rdx, 0

mov rcx, 0

inc r10

#------------------> exit loop max CPUID for Virtual Machine checking Inside looking out

cmp r10, 0x40000010

jle loop

call Exit

#Jonathan David Lee

#Professor Devin Cook