

The x86 Hardware Architecture

Computer Architecture Exploitation and Security

Table of Contents

[Objectives 2](#_Toc41385760)

[Background Reading and Information 2](#_Toc41385761)

[Important Information 2](#_Toc41385762)

[Introduction 3](#_Toc41385763)

[Problem 1 4](#_Toc41385764)

[Problem 2 5](#_Toc41385765)

[Problem 3 5](#_Toc41385766)

[Problem 4 5](#_Toc41385767)

[Problem 5 8](#_Toc41385768)

[Problem 6 8](#_Toc41385769)

L*abs must be submitted by the due date for full credit. After due date late submissions will be accepted for a period of one week (seven days) and the grade will be reduced by ten percent (10%) per day after due day.* ***Assignments that are submitted more than seven days late will receive a grade of zero (0).***

I certify that the work submitted in this assignment is my own and that it has not been taken in whole or in part from any other source. I understand that the penalty for plagiarism will include a grade of zero (0) for this assignment plus disciplinary action in accordance with SAIT policies.

Signature: \_\_\_\_Coleton Sanheim\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

# EVALUATION:

|  |  |  |
| --- | --- | --- |
| Fundamental Data Types | 10 |  |
| AT&T and Intel Assembly Syntax | 5 |  |
| Addressing Mode | 15 |  |
| EFLAGS | 15 |  |
| Inline Assembly - CPUID instruction | 15 |  |
| TOTAL MARK | 60 |  |

Computer Architecture Exploitation and Security

The x86 Hardware Architecture

Objectives

This lab focuses on the following objectives:

* Discuss the advantages and disadvantages of RISC and CISC architectures.
* Explain the interaction among the building blocks.
* Describe byte, word, doubleword and quadword.
* Use basic Intel 64 and IA-32 Instructions

Background Reading and Information

*Intel 64 and IA-32 Architectures Software Developer’s Manual. Volume 1: Basic Architecture*

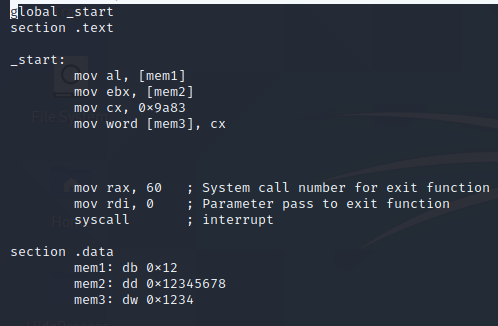
<http://www.intel.com/content/dam/www/public/us/en/documents/manuals/64-ia-32-architectures-software-developer-vol-1-manual.pdf>

Intel Manual Volume 2 -Vol. 2A 3-198

# Problem 1 - Data Types \_\_/10

Read Intel manual Volume 1 section 4.1 (Fundamental Data Types) to differentiate data types defined for Intel 64 and IA-32 architectures.

1. Create the following nasm code and save it as data.nasm



1. Use **ld** to generate the executable
2. Use the respective **gdb** commands to analyze the register values and the content of the addresses **[mem1], [mem2] and [mem3]**
3. Use **; (semicolon)** to comment each uncommented line. **2pts**

**See screenshot below**

1. Explain how placing **mov dl, [mem2]** before the exit code, affect the behavior of the program? **2pts**

**It wouldn’t affect the behavior of the program, it would move the contents of mem2 into the register dl**

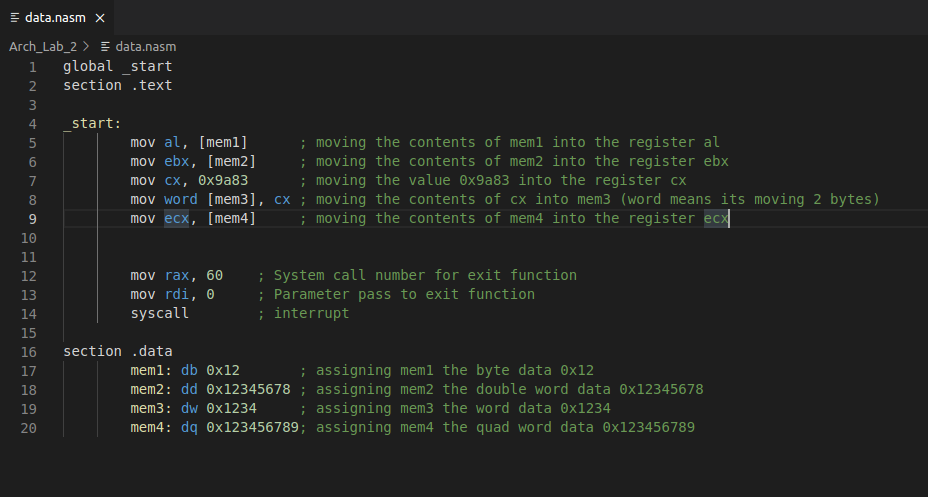
1. Describe what would happened if **mov dword[mem2], dx** is added before exit? **2pts**

**It would give an error of mismatch in operand sizes, because dx is too small to give the required amount of bits for a double word**

1. Modify the code and create a Quadword (dq) data type called mem4. Copy the value store in mem4 into a register of your choice. **2pts**

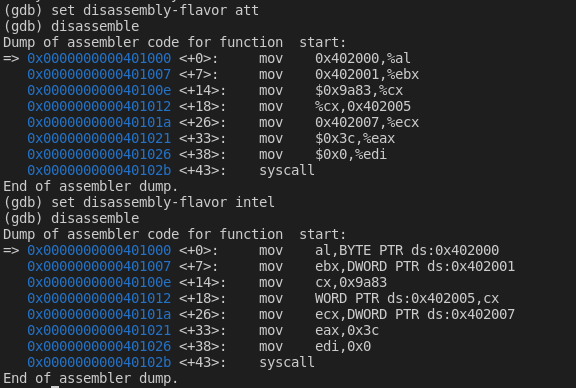
**See screenshot below**

1. Attach screen capture with modified code, commented lines, registers and memory content. **2pts for screenshots**

****

# Problem 2 - AT&T and Intel Assembly Syntax \_\_/5

1. Disassemble the previous code using both the AT&T and Intel format and list three differences in table format. **3pts**
2. **they are reversed between eachother**
3. **intel contains the data type (ie. WORD, DWORD, BYTE)**
4. **intel operands are undelimited where att operands are preceded by $ for immediate, % for registers**
5. Attach the screen capture that demos the disassembly of the program using the two formats **2pts**

****

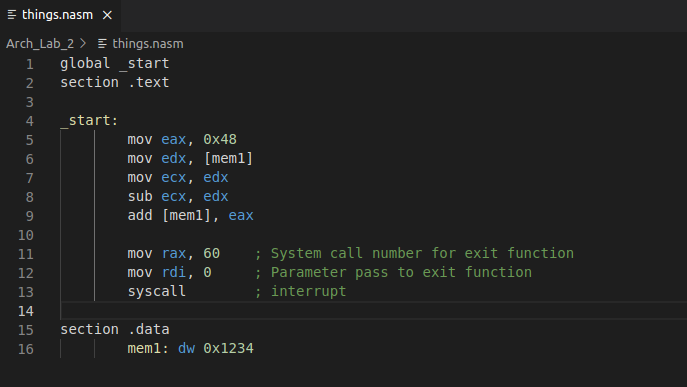
# Problem 3 - Addressing Mode \_\_/15

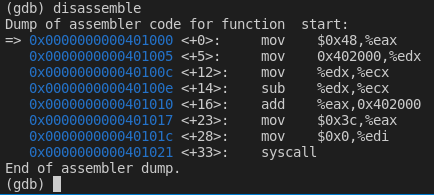
1. Read Intel manual Volume 1 section 3.7.2 “Register Operands “and for MOV instructions details check 2B 435 MOV “Instruction Set Reference” and determine the addressing mode of the following instructions in Intel format:

mnemonic <destination> , <source>

|  |  |
| --- | --- |
| **Instruction** | **Addressing Mode** |
| MOV rbp,rsp | **Move register, register** |
| SUB rsp,0x20 | **Subtract register, immediate** |
| MOV rbx,[rax] | **Move register, memory** |
| ADD rdx,[rcx] | **Add register, memory** |
| SUB [eax], 0x8090 | **Subtract memory, immediate** |
| MOV [rax],rdx | **Move memory, register** |
| MOV [ebx], 0x200 | **Move memory, immediate** |
| MOV eax, 0x3c | **Move register, immediate** |

1. Create nasm code that perform the following instructions: (**7pts**)
2. Move immediate value of 0x48 to register eax
3. Move a value from memory into register edx
4. Move the value from edx register to ecx register
5. subtruct the value in edx from the value in ecx
6. Add value of eax register to a value located in memory
7. Disassembly nasm and analyze the results
8. Attach the screen capture with nasm code
9. Attach screen capture with disassembly results





# Problem 4 - EFLAGS \_\_/15

1. Read Intel manual volume 1 section 3.4.3 EFLAGS Register and section 3.4.3.1 Status Flag
2. Run the following nasm code
3. In the section you will disassemble the code and analyze the registers:
   1. Use Intel manual or lecture slides to differentiate status flags.
   2. Identify the flags that changed when disassembling the code and list them below. **1pts**

**IF**

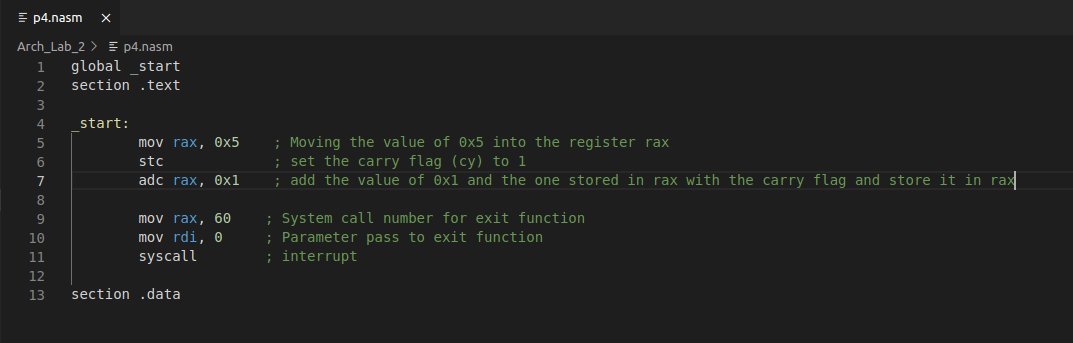
**CF**

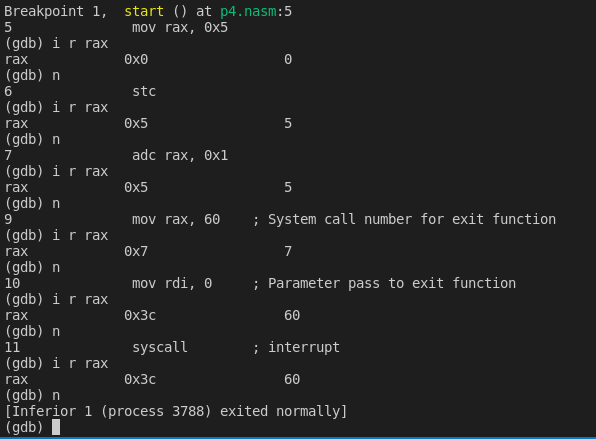
* 1. Search the Intel manual to find the purpose of **stc** and **adc** instructions used in this code. State what those instructions do. **1pt**

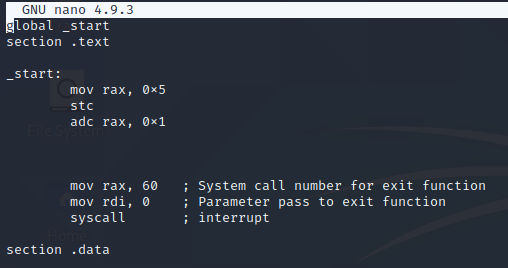
**stc stands for Set the Carry Flag and it sets the cy flag to the 1 state**

**adc stands for Add With Carry and it add the values in Rn and Operand2, together with the carry flag**

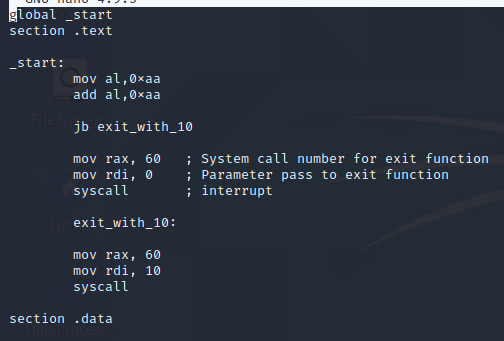
1. Comment the lines providing the main purpose of these instructions. **1pt**
2. Attach the screen capture of documented nasm code and the results **2pts**

****



****

1. One instruction used to control the flow of execution is **jmp** (Jump). It allows the code to be executed in a non-sequential way. Jump can change the flow based on changes in flags of the RFlag register.



1. Run the following nasm code using jump with condition and analyze RFLAG registers
2. Use gdb and identify how RFLAG changes during execution:
   1. Comment lines that are not commented **1pt**
   2. What status flag caused the **JB** instruction to make the jump? **1pt**

**The carry flag, cf**

* 1. Which instruction is executed when the jump is taken? **1pt**

**jb 0x401012 <exit\_with\_10>**

* 1. Modify the code replacing **JB** with **JBE** and **ADD** with **SUB**
  2. How does the **JBE** instruction affect the behavior of the program? **1pt**

**The program still does the jump, but different flags are used**

* 1. What condition(s) would cause **JBE** to perform a jump? **1pt**

**The ZF flag being used, such as with the sub function**

* 1. Attach the commented and modified nasm code with the respective results.
  2. Repeat previous steps to use now **JNS** instead of **JBE** instruction. **2pts**

Now use **SUB** instruction as follows:

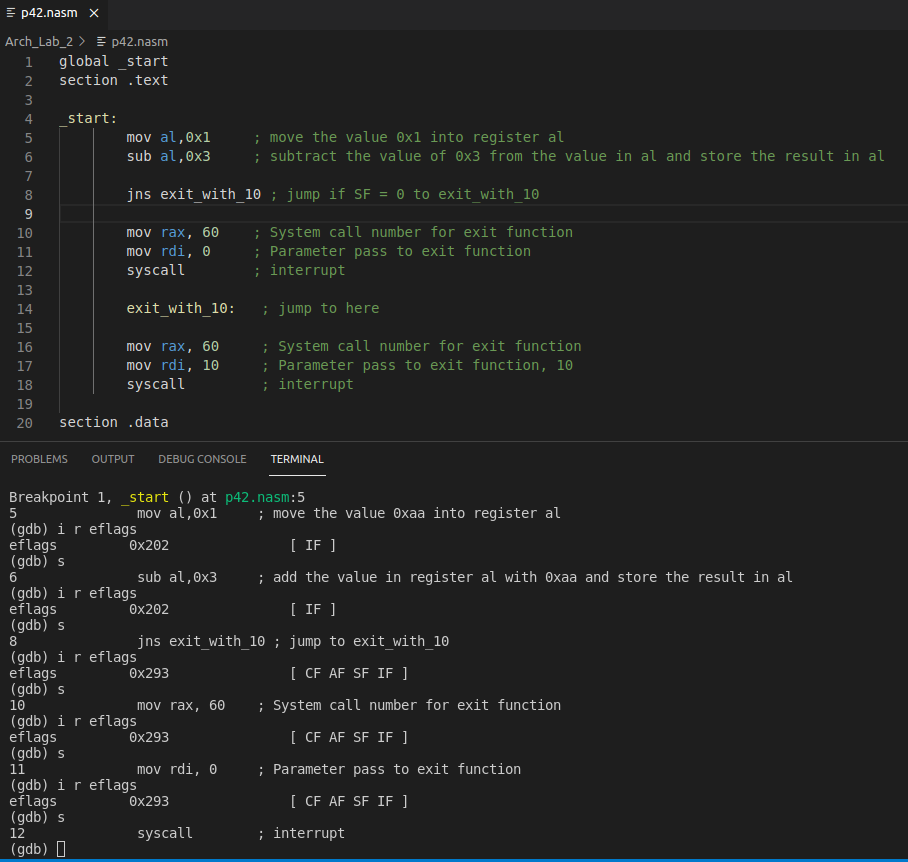
**mov al, 0x1 ; replaces mov al, 0xaa**

**sub al, 0x3 ; replace sub al, 0xaa**

1. Explain how the **JNS** instruction operates, include the flags in your description. **1pt**

**JNS jumps if the flag SF = 0**

1. Attach commented and modified nasm code with respective results. **2pts**



# Problem 5 - CPUID instruction \_\_/5

See page 431 and 792 Volume 2A 3-198 CPUID of the Intel Manual for a description of the CPUID instruction.

The Intel x86 architecture provides instructions that gives the programmer access to the particular processor information. For this problem, find the description of the CPUID instruction in the Intel Architectures Software Developer's Manual***.***

**Describe how you would use the instruction to find information about the size and configuration of the on-chip caches**.

**The approach you could use to answer this question is as follows:**

1. **What registers would you use?**

**You would use EAX**

1. **What values would be placed into those registers?**

**O2H for Cache information**

1. **What instruction would you call?**

**CPUID**

1. **Where would you find the results after completion of the command?**

**In the EAX, EBX, ECX, and EDX registers**

# Problem 6 - Inline Assembly in C \_\_10

See page 431 and 792 Volume 2A 3-198 CPUID of the Intel Manual for a description of the CPUID instruction.

Read the manual to understand the purpose of CPUID instruction and the registers used by this instruction.

Assembly code can be used within C, C++ programming language. When assembly is mixed into your C, C++ program it is called **Inline Assembly**.

Create an Inline Assembly in C, named **m02cpuid.c**, that reports the hexadecimal values in registers EAX, EBX, ECX and EDX.

\_\_asm\_\_ ("cpuid"**:** "=a"(EAX), "=b"(EBX), "=c"(ECX), "=d"(EDX)**:** "0"(request\_code)); // This is inline assembly …**.**

**SEE SCREENSHOT BELOW FOR CODE + RESULT**

