Study Guide

Assembly

Created By: Jose Llerena, Teaching Assistant

Module 1. Basic Assembly

Lesson 1.1: Template and Setup

Skills Learned From This Lesson: What the course is about, Basic requirements, Install packages and create a Hello World program

- Sudo apt install nasm gcc-multilib vim
- To get the package: wget https://cs.unk.edu/~miller/templateMake.tar.gz
- To extract: tar xzf templateMake.tar.gz
- To install: ./templateMake/fix.sh
- Create a project: genMake.sh MyProject
- Vi or Vim can be used to edit text, but recommendation is to use nano
- Open asm file to enter code

Lesson 1.2: Introduction to Assembly

Skills Learned From This Lesson: Architecture, Data Representation

- Second lowest low level language
- It is written for different architectures
- Creates binary code
- The assemblers are MASM, TASM, NASM, GNU
- Some of data representation are Binary and Hexadecimal
- Characters are ASCII or Unicode

Lesson 1.3: Architecture, Registers, and Protected Mode

Skills Learned From This Lesson: Organization, Registers, History and Modes, Paging and Interrupts

- Computer composes of registers, flags, memory addresses, IO function, computing units
- Computer memory: basic unit is the byte = 8 bits, ASCII chars, assembly units
- CPU: Executes Machine Code
- Arithmetic Logic Unit (ALU): Performs Arithmetic Operations
- Floating Point Unit (FLU): Performs floating point math

- Register: high speed, little memory, general purpose
- Flags store info about the previous executed instruction
- Paging: Not all memory is loaded or page is swapped to disk

Lesson 1.4: Binary, 2's Complement and Hexadecimal
Skills Learned From This Lesson: Calculate in binary, decimal and 2's complement, convert to decimal

- Binary uses 1's and 0's
- Hexadecimal uses letters A,B,C,D,E,F
- 2's complement is used by computer to store positive and negative numbers

Lesson 1.5: Assembly Template
Skills Learned From This Lesson: Understand the template,

- The template creates a directory and a make file for each project
- Main files: fix.sh, asm_io.asm, make file, genMake.sh, template.asm

Lesson 1.6: Instructions, Directives and Generating a Listing
Skills Learned From This Lesson: Assembly language, instructions, listing of a file

- Assembler: Approximately 1 to 1 assembly to machine code and machine dependent
- Compiler: High level lines generates many machine/assembly instructions
- NASM: Netwide Assembler
- Operands: Register, Memory Location, Immediate, Implied
- Identifiers are used for vars, constants, procedures or labels
- Listing gives info like offset, binary code for commands

Lesson 1.7: Logical Operators and Memory Layout
Skills Learned From This Lesson: Logical Operators, Memory Hierarchy

- Logical Operators: And, Or, Not, Xor
- Instruction Execution Cycle: Fetch, Decode, Execute, Store
- RISC: Reduced, smaller op codes, more instructions
- CISC: Complex, larger op codes, fewer instructions

Lesson 1.8: Segments and Functions
Skills Learned From This Lesson: Segments, Executing Functions

- Segments: .data, .bss, .text
- Executing Functions: Use call mnemonic to call a function
- Practice with the examples

Lesson 1.9: Sign Extend, Zero Extend, Multiple, Divide

Skills Learned From This Lesson: Changing data sizes, multiplication, division

- Making something smaller means moving to a smaller register
- Converting word to byte means remove upper 8 bytes
- Movsx: Move with sign extension
- Multiplication: mul for unsigned numbers
- Signed multiplication: imul
- Division: div unsigned and idiv for signed

Lesson 1.10: Multiply and Divide Examples

Skills Learned From This Lesson: How to multiply and divide

- Multiplication implicitly uses eax, modifies eax and edx
- Division implicitly uses eax and edx, modifies eax and edx

Lesson 1.11: Compare, Conditionals and Jumps

Skills Learned From This Lesson: Control Structure, Looping

- Control Structure: compare - cmp, branching - jmp
- Looping: loop, loope, loopne
- Check the example

Lesson 1.12: Skeleton and Loop Example

Skills Learned From This Lesson: if translation, looping, and examples

- Jump Zero: jz
- Loop decrements ecx, checks for zero and if not zero jump to top

Lesson 1.13: Shift Left and Right

Skills Learned From This Lesson: Shifting left and right

- Logical shift: move all the bits to the left or the right, last bit moved out set the carry flag, replaced with 0's

Lesson 1.14: Arithmetic Shift

Skills Learned From This Lesson: Double Precision and examples in Arithmetic Shift

- The bits that slide off the end disappear
- Shift arithmetic left -> sal, Shift arithmetic right -> sar
- Double precision shifts: shrd dest, src, cnt, shld dest
- Usage: SHLD/SHRD dest, src, count
- Modifies flags CF, PF, SF, ZF

Lesson 1.15: Module 1 Review

Skills Learned From This Lesson: Review about module 1

- Answer Question: What is a mnemonic and operand
- Answer Question: Smaller version of ebx register
- Answer Question: Names of the 32 bit registers
- Answer Question: What register is used for loop
- Practice Shifting, multiplying and copying data from register to memory

Module 2. Indirect Addressing, Stack, Arrays and Strings

Lesson 2.1: Indirect Addressing and Variables Part 1

Skills Learned From This Lesson: Indirect addressing and variables

- Indirect addressing point to RAM
- Variables are defined in .bss or .data
- Check for string example and notation

Lesson 2.2: Indirect Addressing and Variables Part 2

Skills Learned From This Lesson: Define and modify a string

- Check the example

Lesson 2.3: Stack Intro Part 1

Skills Learned From This Lesson: What is the stack

- Stack is a region of RAM pointed to by the register ESP
- Push and pop things onto and off of the stack
- LIFO operations
- Used mostly for function calls

Lesson 2.4: Stack Intro Part 2

Skills Learned From This Lesson: Stack Operations

- Check for the examples

Lesson 2.5: Stack Usage

Skills Learned From This Lesson: Using the stack and an example calling printf

- Saving Data: Enter, Exit
- Printf: push args on in reverse order

Lesson 2.6: Simple function example

Skills Learned From This Lesson: Create and debug a function

- Label is function name
- Cal instruction calls a function
- Ret instruction returns from a function

Lesson 2.7: Function Prologue

Skills Learned From This Lesson: Function prologue and example code

- Setup entering a function, save the ebp register,
- EBP is extended base pointer

Lesson 2.8: Function Epilogue

Skills Learned From This Lesson: function epilogue and full example

- Check and practice the example

Lesson 2.9: Function Arguments

Skills Learned From This Lesson: How function arguments work

- Allow parameters to be passed to function
- Setup using an ebp based stack
- Restore using epilogue
- Prologue uses ebp to point to the base of our stack

Lesson 2.10: Saving Registers

Skills Learned From This Lesson: Saving register state and saving flags

- Current state of the CPU: When modifying registers or flags in a function
- Push modified, all or flags
- Commands: pushad, popad, pushfd, popfd

Lesson 2.11: More complicated function

Skills Learned From This Lesson: Create function from scratch using parameters, prologue, epilogue, save registers, call external functions

- Check and follow the example

Lesson 2.12: Calling Conventions

Skills Learned From This Lesson: What calling conventions are

- Assumption about how the caller and callee work
- 32 and 64 bit calling conventions

- 32 bit: cdecl, the caller cleans the stack, faster than STD call, parameters are pushed to stack in right to left order
- Calling conventions: stdcall, cdecl, thiscall, fastcall

Lesson 2.13: Local Variables

Skills Learned From This Lesson: How local variables are used

- Used in function
- Value disappears or not accessible when the function is finished
- Local variables use the EBP registers

Lesson 2.14: Local Variables Examples

Skills Learned From This Lesson: Examples of local variables

- Example is about loop to sum numbers and using local variables in main and functions

Lesson 2.15: Enter and Leave

Skills Learned From This Lesson: enter and leave and useful macros

- Is better to use instead of prologue and epilogue
- Leave exits a function

Lesson 2.16: Enter and Leave example conversion

Skills Learned From This Lesson: Convert a program to use macros

- Sum program used hard coded offsets
- Convert to use parameters and arguments

Lesson 2.17: Floating Point

Skills Learned From This Lesson: Floating point numbers and operations

- There is single and double precision
- Floating point unit
- Commands: FLD, FILD, FST, FSTP, FIST, FISTP
- Math operations: FADD, FADDP, FSUB, FSUBP, FMUL, FMULP, FDIV, FDIVP

Lesson 2.18: Floating Point Circle Example

Skills Learned From This Lesson: Example program of a floating point

- Write a program to calculate area of circle and its circumference

Lesson 2.19: Floating Point Comparison

Skills Learned From This Lesson: floating point comparison and additional math instructions

- Fcomp allows to compare floating point numbers
- Commands: FCOM, FCOMP, FCOMPP, FICOM, FICOMP, FIST
- Flags commands: FSTSW, SAHF, LAHF
- Modern instructions: FCOMI, FCOMIP

- Math instructions: FCHS, FABS, FSQRT, FSCALE, FCOS, FSIN, FPTAN

Lesson 2.20: Floating Point Comparison Example

Skills Learned From This Lesson: Example using floating point comparison and comparison gone wrong

- Avoid comparing for equality
- Use greater than or less than or equal to

Lesson 2.21: Max of Three Numbers

Skills Learned From This Lesson: Example of three number max

- Read 3 numbers via scanf
- Return the max of those 3 and uses local variables

Lesson 2.22: Conditional Move

Skills Learned From This Lesson: Pipelines, Order Execution and Conditional Executing

- Pipelines: Allow multiple instruction, fetching and storing are slow
- Branching AKA jumping
- In order processor: fetch instruction, execute and store
- Out of order processor: fetch instruction, add to queue, wait for dependency, executes, store
- Conditional Execution: Ability to execute instructions based on flag, compare and then move data, reduces branching

Lesson 2.23: Conditional Move max example

Skills Learned From This Lesson: Convert the 3 number max program to use conditional execution

- Follow the example in the video

Lesson 2.24: Arrays

Skills Learned From This Lesson: Pointers and arrays

- Pointers: points to a location in RAM
- Load using either mov or lea
- Offset represents how much from the beginning of the array to add

Lesson 2.25: Arrays Examples

Skills Learned From This Lesson: Example using an array

- Example to read a list of numbers, add all the numbers together and print it

Lesson 2.26: String Instructions

Skills Learned From This Lesson: String functions and operations

- Functions: Length, scan instructions (CLD, STD, ESI, EDI), load and store (LODSB, LODSW, LODSD, STOSB, STOSW, STOSD), move / copy (MOVSB, MOVSW, MOVSD), scan string (SCASB, SCASW, SCASD, CMPSB, CMPSW, CMPSD)
- Another functions: Repeat (REP, REPE, REPZ, REPNE, REPNZ)

Lesson 2.27: Strings uppercase examples

Skills Learned From This Lesson: String example

- Example of program to read a string, convert it to uppercase and print it as uppercase

Module 3. ARM ARCHITECTURE

Lesson 3.1: ARM Intro

Skills Learned From This Lesson: Introduction to ARM Architecture

- ARM originally Acorn RISC Machine, later Advanced RISC Machine
- Less transistors, used on mobile phones, less heat
- ARM uses different system modes

Lesson 3.2: ARM Template

Skills Learned From This Lesson: Download and install the template

- Similar to the x86 template
- Follow the example

Lesson 3.3: ARM Math and Data Movement

Skills Learned From This Lesson: ARM Basics - Math and Data Movement

- Operations: ADD, ADC, SUB, RSB, SBC, RSC
- Move data: MOV, MVN
- Loading and storing data: LDR, STR

Lesson 3.4: Branching, if, while, shift

Skills Learned From This Lesson: Branching - If and while, shifting

- *Conditional Branches: EQ, NE, GE, LE, LT, GT, MI, PL, VS, VC, HI, LS, CS, CC/LO*
- *Shifting: Rd, Rm, Rs, Sh, LSL, LSR, ASR, ROR*

Lesson 3.5: Shift Example

Skills Learned From This Lesson: Loop and shift example

- Follow the example in the video

Lesson 3.6: Memory, Offsets, Debugging and Listing

Skills Learned From This Lesson: Memory offsets, Debugging and Disassembly

- Loading Registers: Load and store
- An offset of the PC can be used to store memory addresses and constants

- Debugging tools: GEF, GDB, Strace

Lesson 3.7: Pushing and Popping

Skills Learned From This Lesson: What push and pop are about

- Allows to save the register onto the stack
- Calling functions can destroy registers

Lesson 3.8: Push Example

Skills Learned From This Lesson: Example using stack and saving registers

- Follow the example in the video

Lesson 3.9: Array Indexing

Skills Learned From This Lesson: ARM array and ARM indexing

- ARM Indexing Modes: Pre-Indexed and Post-Indexed

Lesson 3.10: Array Indexing Example

Skills Learned From This Lesson: ARM Array and index example

- Check the example in the video (copy data from 1 string to the other)

Lesson 3.11: ARM Multiple Load and Store

Skills Learned From This Lesson: How to do multiple load and multiple store

- ARM spec: Op, Cond, Rn, Reglist, addr_mode
- Load register module: ldm
- Store Register: stm
- Store multiple: stmia
- Load multiple: ldmia

Lesson 3.12: ARM Load and Store Multiple Examples

Skills Learned From This Lesson: Write an example about ARM load and store multiple

- Problem to solve is to copy string to another buffer and load part of the string into memory

Lesson 3.13: VFP and Neon

Skills Learned From This Lesson: Vector Register and Neon

- Vector Floating Point: Operations, separate co processor, deprecated in favor of neon, uses larger registers
- Neon: Advanced SIMD for ARM, supports multiple data types, parallel processing, Cortex A-8, 64 or 128 bit SIMD operations, can be used for GPS or MP3 decoding
- Neon Instructions: VABS, VADD, VCEQ, VDUP, VLDn, VMAX, VMIN, VMOV, VMUL
- Neon Instructions Vector Load: VLDR, cond, Dd or Sd, constant, datatype
- Neon Instructions Vector Move: Vop, Op, Cond, Datatype, Qd or Dd, Imm

- Neon Instructions Vector Move Long: VMOVL, VMOVN, VQMOVN, VQMOVUN, Q, Cond, Qd, Dm, Dq, Qm
- Neon Vector Multiply: MUL, VMUL, Qd, Qm, Qn, Dd, Dn, Dm

Lesson 3.14: Neon Example

Skills Learned From This Lesson: Example program of Neon, using Neon registers for integer operations

- Problem to be solver: Read to integers, multiply using Neon, print result

Lesson 3.15: Neon Floating Point

Skills Learned From This Lesson: Vector Floating Point and Neon on ARM, VFP and Neon Floating Point Instructions

- Neon and VFP are optional extensions to the ARM Architecture
- VCVT: Between Single Precision and Double Precision

Lesson 3.16: Neon Floating Point Example

Skills Learned From This Lesson: Floating Point Example

- Problem to be Solved: Read radius from user, calculate and print area

Lesson 3.17: SIMD Load and Store Data

Skills Learned From This Lesson: Load and Store in Single Instruction Multiple Data (SIMD)

- Allow processors to operate more efficiently
- Load Data: VST and VLD
- Extension Register Load: VLDR
- Extension Register Store: VSTR

Lesson 3.18: SIMD Process Data

Skills Learned From This Lesson: Process Data of SIMD

- Vector Pairwise Maximum, Vector Pairwise Minimum
- Vector ADD: VADD
- Vector multiply by scalar and accumulate VMLA
- Vector Bitwise Exclusive OR: VEOR

Lesson 3.19: SIMD Encryption Example

Skills Learned From This Lesson: Example of SIMD using Encryption

- Read string from user and read key from user to encrypt data

Lesson 3.20: Thumb Mode

Skills Learned From This Lesson: Thumb Mode and Instructions, changing modes

- ARM Mode: Instructions are 32 bits wide
- Thumb Mode: Instructions are 16 bits wide, code is more compact, fetch and execute

- Changing Mode: BLX - Branch with link and exchange, BX - Branch and Exchange
- Unified Assembler Language (UAL): One sintax for both ARM and Thumb

Lesson 3.21: Thumb Mode Example

Skills Learned From This Lesson: Example Using Thumb

- Problem: Create a program to print a tree of *'s and write a function to print a certain number of *'s

Lesson 3.22: Conditional Execution

Skills Learned From This Lesson: Flags and Instructions in Conditional Execution

- Allows an instruction to be executed only when a condition is met
- Instruction updating flag: ADD - add without a carry
- Flags: Negative -> N, Zero -> Z, Carry -> C, Overflow -> V

Lesson 3.23: Conditional Execution Example

Skills Learned From This Lesson: Solving a problem using conditional execution

- Problem to solve: write a program to get a random number and guess it

Lesson 3.24: IT Block Assembly

Skills Learned From This Lesson: IT blocks in conditional execution

- Thumb-2 conditional execution: does not have the bits for stand alone conditional instructions
- IT blocks available for both Thumb-2 and ARM mode
- IT Blocks allows up to 4 instructions to be conditionally executed
- ITxyz FLAG: xyz can be T->Then or E->Else

Lesson 3.25: IT Block Example

Skills Learned From This Lesson: Thumb conditional execution example

- Problem to be solved: Create a min/max functions using IT blocks

Module 4. C Constructs and Interrupts

Lesson 4.1: Tools for Code, Reverse Engineering

Skills Learned From This Lesson: Compilation process and tools

- Examples of compilers: GCC, Visual Studio, Mingw
- Compiler Process: Take high level program, generates an intermediate representation, apply optimizations and generate an output program
- Output file format: binary file format, readable by OS
- Binary format readers: PEView, objdump, disassembler
- Disassembler: Objdump, IDA Pro, Binary Ninja, Ghidra

- Hex Editor: Allows to view, edit and save files using hex representation

Lesson 4.2: Reverse Engineering Process

Skills Learned From This Lesson: RE Process and simple example

- Disassembler is a computer program that translates machine language into assembly language
- Disassembly process: Reads in binary data, determine layout and references, instructions
- Reading binary data: Use magic numbers to guess at format, understand the OS method of loading the binary; determine offsets, architecture and OS regarding memory locations
- Determine layout and references: Entry point, dynamic libraries, and system calls
- Disassemble instructions: start at entry point, look up the instruction length, disassemble instruction, determine if a branch occurs, continue until all instructions in the queue are processed
- Types of disassemblers: Linear sweep, recursive descent

Lesson 4.3: Setup Reverse Engineering Lab

Skills Learned From This Lesson: Methods to reverse programs online and local

- Online reversing: Online Disassembler, Compiler Explorer
- Setup: Determine target OS, install required compiler and disassembler
- Process: Edit and compile code, sync to disassembler, disassembler, repeat steps

Lesson 4.4: Structures and Unions

Skills Learned From This Lesson: Overview of C constructs and examples

- Constructs provide features to the programmer, allow programs to be created more efficiently

Lesson 4.5: Structure Layout

Skills Learned From This Lesson: Structure of layouts and offsets

- Structure layout: Starts at first declaration, round to the nearest even byte boundary for the next size

Lesson 4.6: Structure Creation: Reverse Engineering

Skills Learned From This Lesson: Using structures

- Example: program that reads in the time and prints out

Lesson 4.7: Structures, Unions and Malloc

Skills Learned From This Lesson: Memory allocation, structures and unions using malloc

- Memory allocation: allocates memory, dynamically allocates memory, give back to the heap

- Heap and stack use free memory and when they collide there is an error

Lesson 4.8: Structures, Unions and Malloc Example

Skills Learned From This Lesson: Example using malloc in assembly

- Example using structures and malloc: get the real time from the clock_gettime system call and use the same structure as before, leverage malloc

Lesson 4.9: Jump Tables and Switch Statements

Skills Learned From This Lesson: Switch statement in C, jump tables

- Switch statement: allows a programmer to easily choose a block of code to execute, easier to read syntax than if-else blocks, based on discrete cases, provides a default case
- Compiler determines when to build a jump table
- Jump table: an array of pointers to program code, selection of a pointer is calculated using math

Lesson 4.10: Jump Table Example

Skills Learned From This Lesson: Create a jump table in assembly

- Define data in code section
- Example is to calculate a schedule based on day of month

Lesson 4.11: Function Pointers

Skills Learned From This Lesson: How function pointers work

- Points to a function or subroutine, allows the functionality to be changed at runtime

Lesson 4.12: Function Pointers Example

Skills Learned From This Lesson: Create and use a function pointer in assembly

- Problem to be solved: Write assembly that reads a number, if number is 1 multiply second input by 2, if is not 1 (else) multiply second input by 8

Lesson 4.13: Inline Assembly

Skills Learned From This Lesson: Inline Assembly, extra arguments, inline function call

- GNU Compiler Collection (GCC) uses AT&T syntax for x86/x64 code
- Variables substitution: variables can be passed in via symbolic names

Lesson 4.14: Inline Assembly Example

Skills Learned From This Lesson: Example of Inline Assembly

- Problem to solve: Capture the time using the Time Stamp Counter, save to a qword and print the result

Lesson 4.15: Assembly with C

Skills Learned From This Lesson: Using Assembly and C together

- GCC and NASM are going to be used
- Example about using a string length function

Lesson 4.16: SysCall and Interrupts

Skills Learned From This Lesson: How Interrupts and SysCall work

- Interrupts are events that signal to the CPU that something occurred
- Type of interrupts can be hardware, software and exceptions
- System Calls provide a way to interact with the OS kernel

Lesson 4.17: Interrupts Example Use Fork

Skills Learned From This Lesson: Write system calls using int 80

- Problem to solve: Use the fork system call to create 2 processes and each prints right before it finishes
- There is also a Wait which waits for a process to finish

Lesson 4.18: Strings in C

Skills Learned From This Lesson: Strings in C in Assembly on the stack, heap and data

- C Strings: C allows string to allocated several different ways

Lesson 4.19: Integers in Assembly

Skills Learned From This Lesson: Standard Types; Integers, Long, Short, bytes in C

- Integers: Signed, Unsigned, DWORD size, whole numbers
- Shorts: Size of 16 bits, DWORD data movement
- Long: Provides 4-8 byte data types, depends on the architecture

Lesson 4.20: 64-Bit Assembly

Skills Learned From This Lesson: 64bit assembly, register, deprecated functions, calling functions

- 64 bit references the number of bits used for memory addresses, registers and stack operations are adjusted
- Processor cores: Multiple CPUs per processor, core counts
- Registers: R prefix
- Deprecated Instructions: PUSHAD, POPAD, library functions will not work asm_io.inc
- Calling Functions: FastCall is used for newer OS

Lesson 4.21: MMX, SSE, AES-NI

Skills Learned From This Lesson: SSE and MMX extensions, AES-NI extensions

- MMX stands for MultiMedia eXtension
- SSE stands for Streaming SIMD Extensions which is of common use in multimedia
- AES-NI stands for Advanced Encryption Standard New Instructions

Lesson 4.22: AES Implementations
Skills Learned From This Lesson: Cross Platform AES, Intrinsic Implementation and disassemble AES intrinsics

- Investigate about the benefits of using AES-NI
- Pay special attention to the examples to have a good understanding

Lesson 4.23: Implement Dump Registers
Skills Learned From This Lesson: Write a 64 bit function

- Problem to solve: write a function to dump the registers

Lesson 4.24: Static and Dynamic Linking Assembly
Skills Learned From This Lesson: Static linking, Dynamic linking -> GOT, PLT

- Static Linking: Provides all code within a binary, uses more space, multiple programs that use the same library
- Dynamic Linking: Linking library code at runtime, address space layout randomization (ASLR)
- Global Offset Table (GOT): A table of addresses that point to dynamic addresses, initially contains address of dynamic linker, updated by the dynamic linker to get actual offset
- Procedure Linkage Table (PLT): Provides a mechanism to resolve dynamic memory addresses, allows code to be PIC and PIE, determines the path to look for libraries

Lesson 4.25: Shared Library
Skills Learned From This Lesson: How shared library works

- Create using flags to gcc: -fPIC, -shared (name with .so)
- Creating main: -I -> header search path, -L -> directory to search for shared code, -la -> library to add

Lesson 4.26: Shared Library Example
Skills Learned From This Lesson: Write a shared library in assembly

- Problem to solve: Write code and methods in assembly, use C method calls to setup writable and executable memory, copy assembly bytes to memory. Use a function pointer to execute memory in buffer