Assignment-3 Concepts on Architecture and Assembly

Full Credit: 20 pts | Due: Sunday 3/6

Important note: ***You MUST read corresponding sections and understand these concepts first, then use your own language (by rephrasing) answering these questions. You MUST NOT copy the lines verbatim (word-by-word) from the textbooks or the Internet!***

Q1. Computer architectures and components

1) Explain these terms (abbreviations) and their definitions: ISA, RISC, CISC, RAM.

RAM (random access memory) - is like storage but way faster then hard drive. And it is volatile (it can store data as long as it powered, so when power is off all data stored in ram are lost. That is why it can't be used for long storage, despite being the fastest storage. It stored in motherboard's memory slots. In order program to run, it need to be loaded to RAM first. So when program stored in hard drive, when we run it, it goes to RAM where CPU can execute it this program. If it is not enough RAM then it will do extra work and load from hard drive, which it will slow computer down, so need increase amount of RAM. And result is faster computer. RAM consists of capacitors (it is like bucket of electricity which hold 0 and 1, but they can not hold charge for long without electricity, they constantly leak)

ISA (instruction set architecture) is machine language that is visible to developer. It gives commands to CPU and tells machine what to do.

RISC (reduced instruction set computer) - take advantages that generally instructions for computer quite simple and computer designed to handle those instructions quickly.

CISC (complex instruction set computer) - because of complex instruction set, can process complex code more quickly.

4) Explain these terms and their functionalities: ALU, register, program counter (PC), and instruction register (IR).

ALU (arithmetic logical unit) receive information (bits) through registers, do calculation and store them back in register. Based on information from Control Units bits, ALU knows which information to perform.

A register is temporary storage build in CPU for faster access.

Program counter points to memory address of instruction and it store to instruction register. If it is big instruction (not finish) CPU can not execute it, so it repeat cycle: fetch-decode-execute and every time PC increment by 1. After execution is completed it removed from Instruction Register. PC goes to next instruction and repeat cycle.

Q2. Instruction execution cycle (fetch-decode-execute-store)

1) What actions happen during the fetch stage of an instruction execution cycle? Is PC incremented at the beginning or end of the fetch stage? Is IR being updated?

The instruction is retrieved by CPU from main memory. PC incremented at the beginning of the fetch stage and IR is being updated.

2) What happen during the decode stage of an instruction execution cycle?

The instruction is broken down into smaller components to determine what instruction is

3) What happen during the execute stage of an instruction execution cycle?

Instruction is executed. Data may be read/write from/to main memory.

4) What happen during the store stage of an instruction execution cycle?

The output is stored in a register.

Q3. Assembly language and ARM64 / ARMv8

1) Why do we need to learn assembly language and why do we choose ARM64?

It helps to understand how computers work under the hood, which make us better decision as software engineer. It helps to understand why something slower than others.
ARM gives you one task at a time and you would do it and come back to get next one and so on. While x86 gives you all task and you figure out how to get them all done. ARM follows RISC instruction set.

2) How many general purpose registers are there in ARM64 CPU?

The thirty one general purpose registers in the main integer register bank are named R0 to R30, with special register number 31 having different names, depending on the context in which it is used.

What are their names and sizes?

Argument registers (X0-X7) 32 bits, Caller-saved temporary registers (X9-X15) 32 bits, Calle saved register (X19-X29) 32 bits, Register with special purpose X8 indirect result register 32 bits, X16, X17 temporary registers 32 bits, X18 platform register 32 bits, X29 pointer register 32 bits, X30 link register 32 bits.

How can they be used as 32-bit registers?

When the registers are used in a specific instruction form, they must be further qualified to indicate the operand data size (32) – and hence the instruction's data size.

3) What model does ARM follow for memory access? How does that model work?

Aside from exclusive and explicitly ordered loads and stores, addresses may have arbitrary alignment unless strict alignment checking is enabled (SCTLR.A==1). However if SP is used as the base register then the value of the current stack pointer prior to adding any offset must be quadword (16 byte) aligned, or else a stack alignment exception will be generated. A memory read or write generated by the load or store of a single general-purpose register aligned to the size of the transfer is atomic. Memory reads or writes generated by the non-exclusive load or store of a pair of generalpurpose registers aligned to the size of the register are treated as two atomic accesses, one for each register. In all other cases, unless otherwise stated, there are no atomicity guarantees.

4) What is an addressing mode?

The term addressing modes refers to the way in which the operand of an instruction is specified.

What are the addressing modes in ARM64? Explain them with examples.

ARM64 follow T32, using a 64-bit base address from a general register Xn (n=0-30) or the current stack pointer SP, with an immediate or register offset. Base plus offset addressing means that the address is the value in the 64-bit register base plus an offset. Pre-indexed addressing means that the address is the value in the 64-bit register base plus offset, then the address is written back to base. Post-indexed addressing means that the address is the value in the 64-bit register base, then address plus offset is written back to base. Literal addressing meads that the address is the value in the 64-bit program counter PC plus a 19-bit signed word offset, i.e. a word-aligned address within ±1MiB of PC. Only available for loads of 32 bits or larger and prefetch instructions: PC is not usable in other addressing modes. The syntax for label is described in section 5 below. An immediate offset may be unsigned or signed (two's complement), and unscaled or scaled, depending on the type of load/store instruction. When scaled it is encoded as a multiple of the transfer size, but the assembly language always uses a byte offset with the assembler/disassembler converting as necessary. The usable byte offsets therefore depend on the type of load/store instruction and the transfer size.

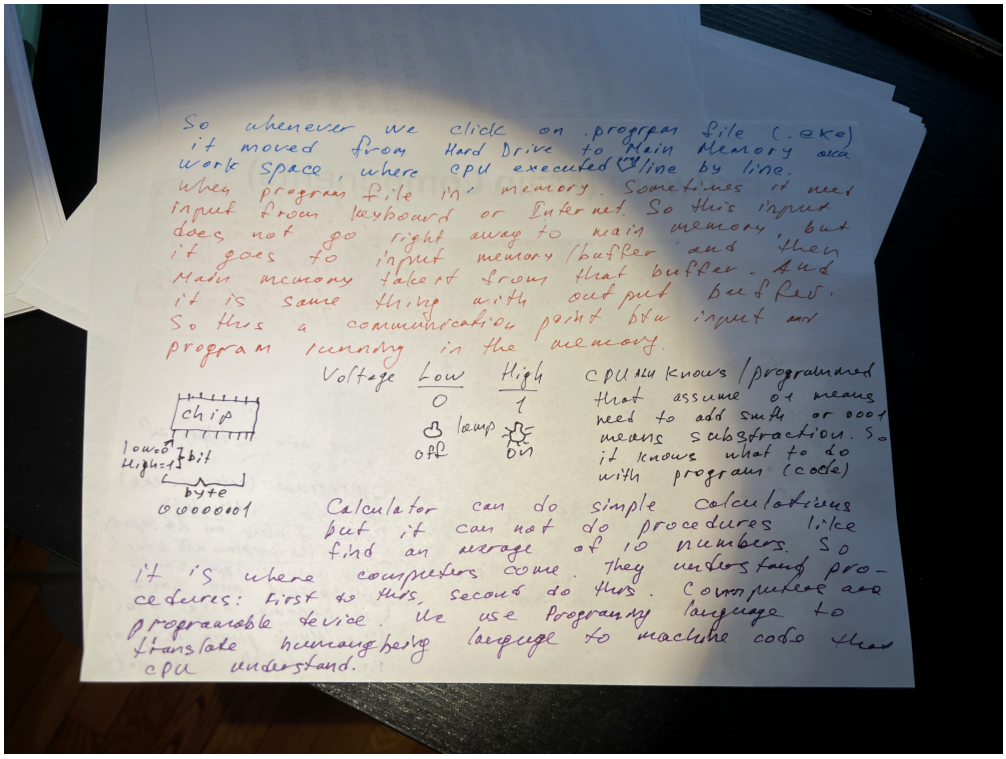
Q4. Call stack and stack frame

1) What is the call stack? What is a stack frame? Explain both and their relationship.

When function is called, operating system set aside space in memory for function to do its job. And this memory call stack frame. When there are n-amount of functions system set aside n-amount of stack frames but only the most resent one is active. When the most recent do it is job like return a value, it is disappear from stack ("last in, first out") and then it goes down to frame below and repeat same thing till no frames left in stack.

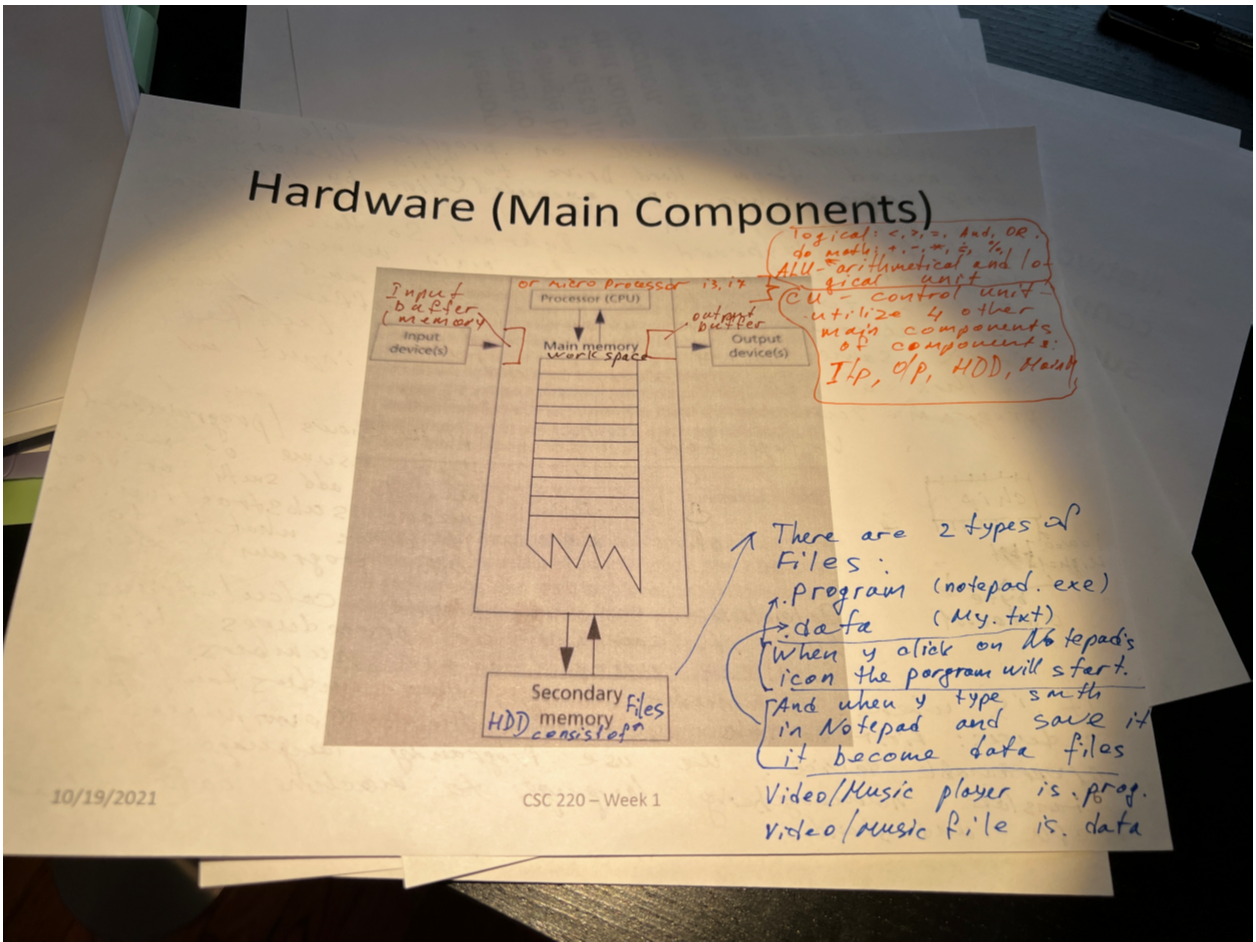
2) What are stack pointer and frame pointer? Explain both and their relationship?

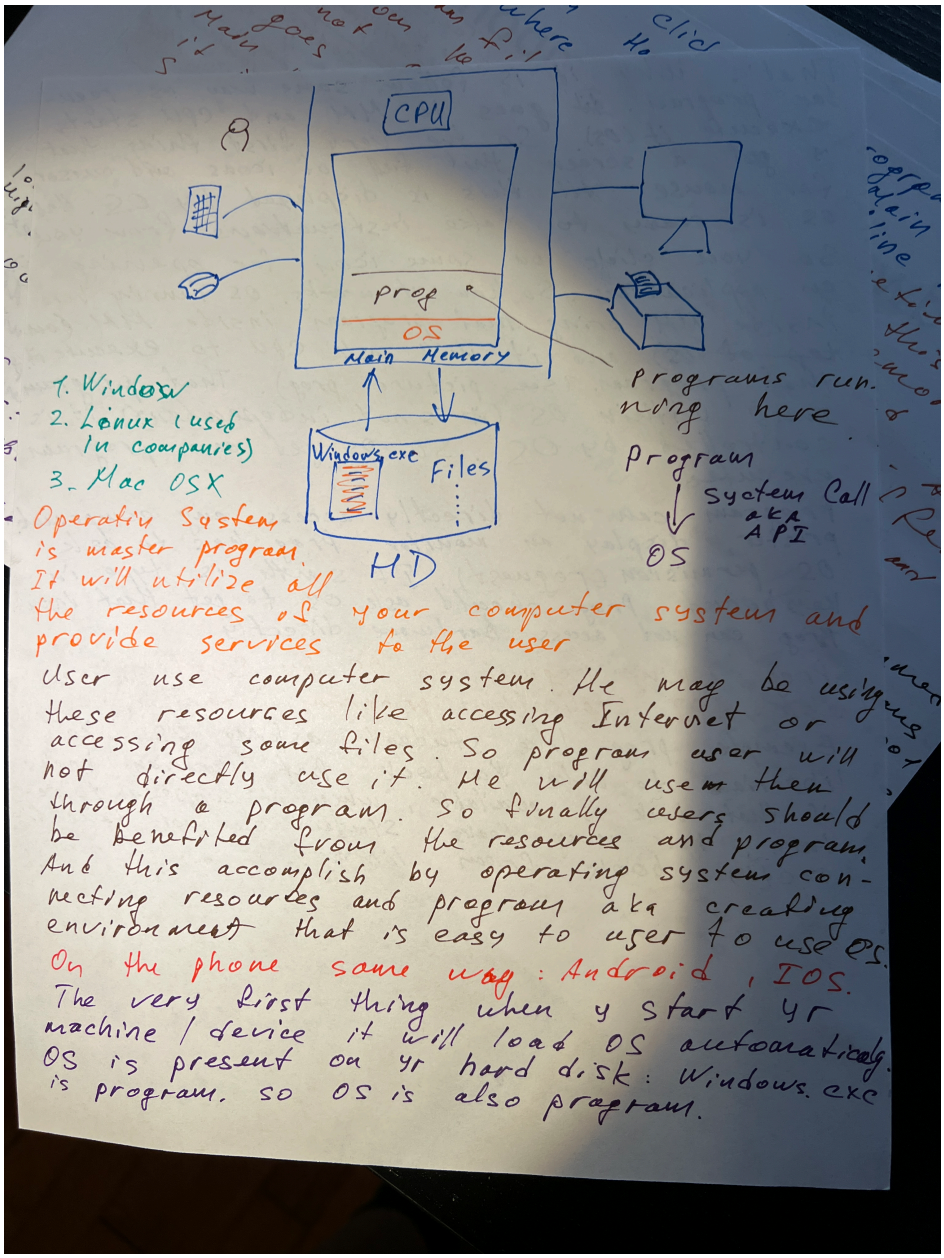
Stack pointer points to last item in the stack or next place to put an item in stack. Frame pointer is static. Stack pointer points to top of the stack and FP points to the bottom of current stack frame.

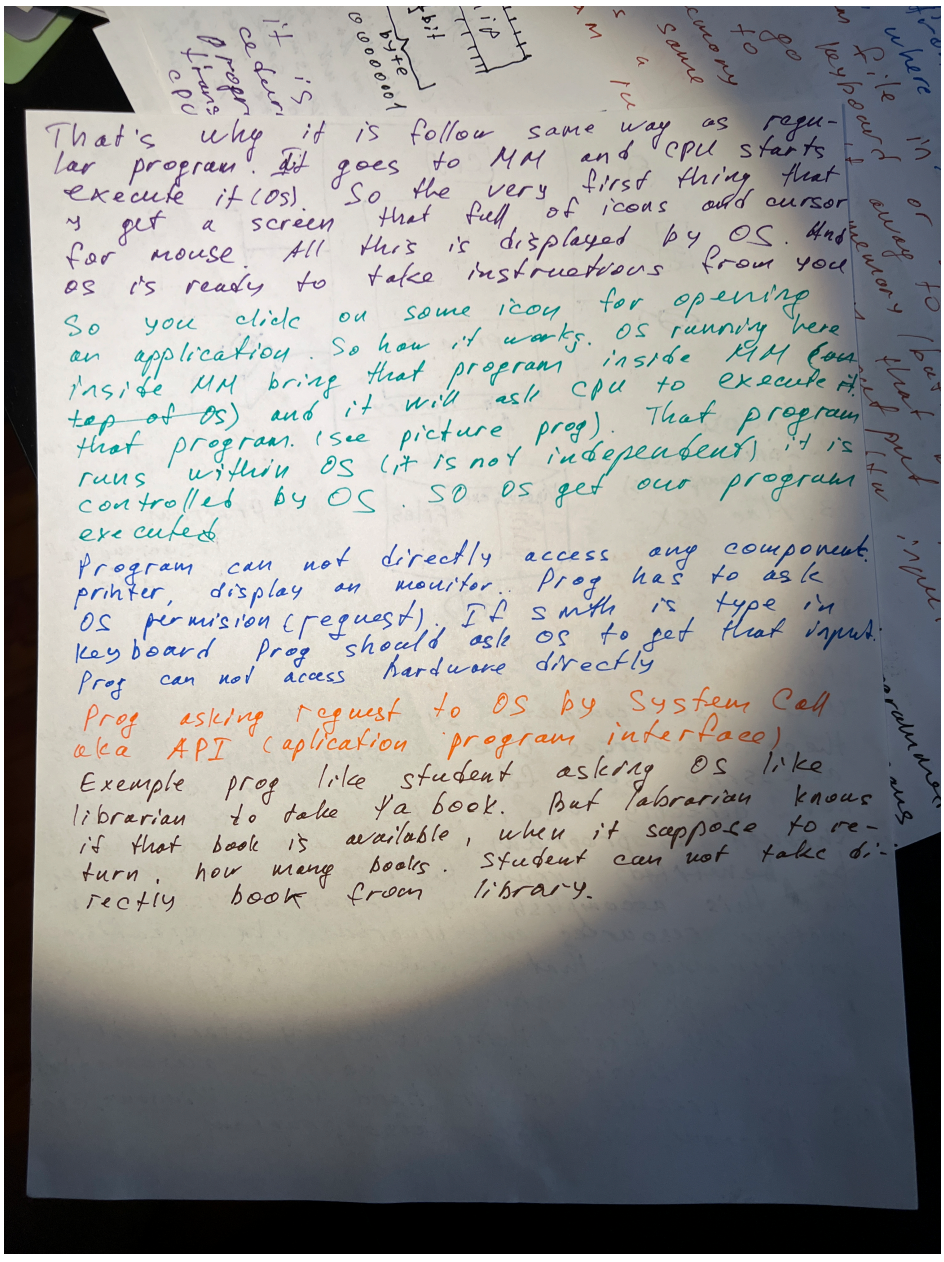


3) How does the call stack grow when a function is called in ARM64 programs?
It grows from lower address to higher and vice versa

2) What are von Neumann architecture and Harvard architecture? What are their differences?
What are the five units and their functionalities of von Neumann architecture?




CPU
prog
OS
Main Memory
Windows.exe
Files
HD
1. Windows
2. Linux (used in companies)
3. Mac OSX
Programs running here.
Program
System Call aka API
OS
Operating System is master program. It will utilize all the resources of your computer system and provide services to the user
User use computer system. He may be using these resources like accessing Internet or accessing some files. So program user will not directly use it. He will use them through a program. So finally users should be benefited from the resources and program. And this accomplish by operating system connecting resources and programs aka creating environment that is easy to user to use OS.
On the phone same way: Android, IOS.
The very first thing when y start yr machine / device it will load OS automatically. OS is present on yr hard disk: Windows.exe is program. so OS is also program.



That's why it is follow same way as regular program. It goes to MM and CPU starts execute it (OS). So the very first thing that y get a screen that full of icons and cursor for mouse. All this is displayed by OS. And OS is ready to take instructions from you

So you click on some icon for opening an application. So how it works. OS running here inside MM bring that program inside MM (on top of OS) and it will ask CPU to execute that program. (see picture prog). That program runs within OS (it is not independent) it is controlled by OS. SO OS get our program executed

Program can not directly access any component printer, display on monitor.. Prog has to ask OS permision (request). If smth is type in keyboard prog should ask OS to get that input. Prog can not access hardware directly

Prog asking request to OS by System Call aka API (aplication program interface)

Exemple prog like student asking OS like librarian to take ya book. But librarian knows if that book is available, when it sapposse to return, how manny books. Student can not take directly book from library.

In Harvard architecture is two memory.

Submission: Place all your answers into a single file, name it as **LastFirstInitial_a3.pdf** (e.g., if your name is John Smith, it will be SmithJ_a3.pdf), and then upload it through your blackboard account by the due day. Note: if you cannot write into the pdf directly, then you can work on your questions in Microsoft Word and then save it as pdf before submission.