**Please explain how the stack is used for subroutine call and return (Chapter 7 “Stack Instructions” (Maximum 1 point)**

**You may describe figure 7.14 “Fixed Location Subroutine Address Storage” posted on page 221 (5th Edition”) You may also use any Internet Resources (don’t forget to use the APA format References). Your response to this question should be substantial (minimum 100 words). The explanations should be written (without copy and paste any graphs or diagrams). Write clearly so that anyone interested in the explanation can understand it.**

***Answer:***

**A subroutine is a set of instructions which are used repeatedly in a program. I**n computer programming, a subroutine is a sequence of program instructions that performs a specific task, packaged as a unit. Subroutines are blocks of code that may be repeatedly called by the main program to serve a given function. Subroutines are executed during program run time. Following is a data transfer subroutine with the name of loadAB, which loads the accumulators ACCaHI, ACCaLO, ACCbHI, and ACCbLO with 01FFh and 7FFF, their respective values. This subroutine has a label of loadAB and ends with a return. The call instruction places the PC + 1 value in the stack. The return instruction pops the PC + 1 value off the stack and returns control to the main program.

Now, considering the figure 7.14 it makes call to first to the stack memory. While there is subroutine program already stored which will be executed by the program given in the figure. In the next figure it makes call “after subroutine”. In the first figure the instruction makes call from 55 to 70 subroutine. In the next figure it calls after subroutine so it calls instruction after the subroutine so mainly, subroutine re-called from 75, within the subroutine.

Stack is a way of computer to store a data. It is One of the most important data storage structures in programming. It stores most recent data in the top therefore, it is also known as LIFO (last in first out). Stack is a basic data structure which can be implemented anywhere in the memory. It can be used to store variables which may be required afterwards in the program Execution. In a stack, the first data put will be last to get out of a stack.

Stack is also important for storing the return addresses and arguments from subroutine calls. Storage or retrieval of words in the stack memory area is the same as accessing items from a stack of items. The return address is the place in the calling program that is returned to when subroutine exits.

Now, considering the figure 7.14 it makes call to first to the stack memory. While there is subroutine program already stored which will be executed by the program given in the figure. In the next figure it makes call “after subroutine”. In the first figure the instruction makes call from 55 to 70 subroutine. In the next figure it calls after subroutine so it calls

**Part 3. Explain a computer's register-level architecture, including (Maximum 1 point)**

**a) CPU-memory interface (Your response to this question should be substantial (minimum 100 words). The explanations should be written (without copy and paste any graphs or diagrams)**

**b) special-use registers (minimum 100 words). The explanations should be written (without copy and paste any graphs or diagrams).**

**c) addressing modes (minimum 100 words). The explanations should be written (without copy and paste any graphs or diagrams).**

*Answer:*

Computer Architecture have many possible designs, the new design there are many possibilities. Such as the large number of registers and register-to-register instructions that are characteristic of newer architectures. A processor register is a quickly accessible location available to a digital processor's central processing unit (CPU). Registers usually consist of a small amount of fast storage, although some registers have specific hardware functions, and may be read-only or write-only. Registers are typically addressed by mechanisms other than main memory, but may in some cases be memory mapped. the improvement possible by using multiple data registers to implement an ADD instruction. Since the register-to-register add can be done directly, the number of steps in the cycle is reduced from four to three, with only a single execute step, and the extra time required for the memory access is eliminated.

1. CPU-memory interface: The fetch–execute CPU implementation reduces instruction fetch delays with modern instruction and branch control technologies. The use of register-to-register instructions also reduces delays. Nonetheless, memory accesses are always required to move the data from memory to register and back, and improvements in memory access still have an impact on processing speed. The memory in modern computers is usually made up of dynamic random access memory circuit chips. DRAM is inexpensive. Each DRAM chip is capable of storing millions of bits of data. Dynamic RAM has one major drawback, however. With today’s fast CPUs, the access time of DRAM, known as memory latency, is too slow to keep up with the CPU, and delays must be inserted into the LOAD/STORE execution pipeline to allow memory to keep up. Within the instruction fetch–execute cycle, the slowest steps are those that require memory access. Therefore, any improvement in memory access can have a major impact on program processing speed. Memory interfacing is an essential part in computer system design. In fact, the among silicon area devoted to memory in a typical digital embedded system or a computer system is substantial. The control signals go between the two to control the transfer of information, and is in general governed by the microprocessor which acts as the master.
2. Special-use registers: The execution unit contains the arithmetic/logic unit and the portion of the control unit that identifies and controls the steps that comprise the execution part for each different instruction. The ALU provides the usual computational abilities for the general registers and condition flags. There is a bus interface unit that provides the logic and memory registers necessary to address memory over the bus. The number of instructions held will depend upon the size of each instruction, the width of the memory bus and memory data register,1 and the size of the buffer. As instructions are executed, the fetch unit takes advantage of time when the bus is not otherwise being used and attempts to keep the buffer filled with instructions. Special use of registers mainly includes the program counter, stack pointer, and status register. In embedded microprocessors, they can also correspond to specialized hardware elements.
3. Addressing modes: A way in which an operand of an instruction is specified is known as addressing modes. Information contained in the instruction code is the value of the operand or the address of the result or operand. Following are the main addressing modes that are used on various platforms and architectures. A single CPU might provide a number of different variations to increase the flexibility of the instruction set. This flexibility also includes the ability to code programs that process lists of data more efficiently. The various ways of addressing registers and memory are known as addressing modes. There are seven types of addressing modes:
4. Immediate Mode: The operand is an immediate value is stored explicitly in the instruction.
5. Index mode: The address of the operand is obtained by adding to the contents of the general register a constant value.
6. Indirect mode: The address of the operand is the content of the main memory or register whose address appears in the instruction.
7. Absolute mode: The address of the operand is embedded in the instruction code.
8. Register mode: The name of the CPU register is embedded in the instruction and the register contains the value of the operand.
9. Displacement mode: Base register contains a pointer to a memory location. An integer is also referred to as a displacement.
10. Autoincrement /Autodecrement Mode: A special case of indirect register mode. The register whose number is included in the instruction code, contains the address of the operand

Reference:

Englander, I. (2014). The \*architecture of computer hardware, systems software, and networking: An information technology approach: An information technology approach. In The \*architecture of computer hardware, systems software, and networking: An information technology approach: An information technology approach (pp. 221). Hoboken: Wiley.

Englander, I. (2014). The \*architecture of computer hardware, systems software, and networking: An information technology approach: An information technology approach. In The \*architecture of computer hardware, systems software, and networking: An information technology approach: An information technology approach (pp. 236-255). Hoboken: Wiley.