****Name:Xunzhi Li****

****ID:10457500****

****Part 2. Please explain how the stack is used for subroutine call and return (Chapter 7 “Stack Instructions” (Maximum 1 point)****

As an analogy, stacks are frequently described by the way plates are stored and used in a cafeteria. New plates are added to the top of the stack, or *pushed*,and plates already on the stack move down to make room for them. Plates are removed from the top of the stack, or *popped*, so that the last plates placed on the stackare the fifirst removed. Similarly, the last number entered onto a computermemory stack will be the first number available when the stack is nextaccessed. Any data that must be retrieved in reverse order from the way it wasentered is a candidate for the use of stacks.

Stacks are also an excellent method for storing the return addresses and arguments from subroutine calls. Program routines that are recursive must “call themselves”. Suppose the return address were stored in a fifixed location.The return address is stored on a stack. This time when the routine is again called, the original address is simply pushed down the stack, below the most recent address. Notice that the program “winds its way back out” in the reverse order from which the routines were entered.

Cited: *Englander, Irv*. (2014). *The Architecture of Computer Hardware and System Software An Information Technology Approach*. John Wiley & Sons.

****Part 3****. ****Explain a computer's register-level architecture, including****

**a)CPU-memory interface**

The most critical interface in any computing system is the connection between memory and the CPU. If this interface doesn’t function properly, the CPU cannot function because it cannot retrieve instructions. If the processor can’t reliably retrieve instructions, it really doesn’t matter if anything else on the board works — you won’t be using it anyway.Understanding the CPU/memory interface is important to more than just the data and instruction stream. In most systems, peripherals share the data and address buses with memory. Thus, understanding the protocol for these buses is important to understanding much of the hardware.

Cited: *<http://www.embeddedlinux.org.cn/EmbSysFirmDemy/0015.html>*

****b) special-use registers****

A Special Function Register (or Special Purpose Register, or simply Special Register) is a [register](https://en.wikipedia.org/wiki/Processor_register" \o "Processor register) within a [microprocessor](https://en.wikipedia.org/wiki/Microprocessor" \o "Microprocessor), which controls or monitors various aspects of the microprocessor's function. Depending on the [processor architecture](https://en.wikipedia.org/wiki/Comparison_of_CPU_architectures" \o "Comparison of CPU architectures), this can include, but is not limited to:

* [I/O](https://en.wikipedia.org/wiki/Input/output" \o "Input/output) and peripheral control (such as serial ports or general-purpose IOs)
* timers
* [stack](https://en.wikipedia.org/wiki/Call_Stack" \o "Call Stack) pointer
* stack limit (to prevent overflows)
* [program counter](https://en.wikipedia.org/wiki/Program_counter" \o "Program counter)
* [subroutine](https://en.wikipedia.org/wiki/Subroutine" \o "Subroutine) [return address](https://en.wikipedia.org/wiki/Return_statement" \o "Return statement)
* processor status (servicing an interrupt, running in protected mode, etc.)
* condition codes (result of previous comparisons)

Because special registers are closely tied to some special function or status of the processor, they might not be directly *writeable* by normal instructions (such as adds, moves, etc.). Instead, some special registers in some processor architectures require special instructions to modify them. For example, the program counter is not directly writeable in many processor architectures. Instead, the programmer uses instructions such as return from subroutine, jump, or branch to modify the program counter. For another example, the condition code register might not directly writable, instead being updated only by compare instructions.

Cited: *<https://en.wikipedia.org/wiki/Special_function_register>*

****c) addressing modes****

As we mentioned previously in our discussion of memory size, an effective alternative to large instructions or variable instruction words is to store the address that would otherwise be located in an instruction word address fifield at some special location that can hold a large address, such as a general-purpose register, and use a small address fifield within the instruction to point to the register location. There are a number of variations on this theme. This technique is used even on systems that provide variable length instructions. A single CPU might provide a number of different variations to increase the flflexibility of the instruction set. This flflexibility also includes the ability to code programs that process lists of data more effificiently. The various ways of addressing registers and memory are known as addressing modes. The Little Man Computer provides only a single mode, known as direct addressing. The alternative just escribed is called register-deferred addressing

Cited: *Englander, Irv*. (2014). *The Architecture of Computer Hardware and System Software An Information Technology Approach*. John Wiley & Sons.