**Assembly 2 Project**

Pawan Kumar

Department of Computer Science, Stevens Institute of Technology

CPE 550-A, Computer Organization and Programming

Dr. Edward Banduk

November 1st, 2021

**Assembly 2 Project**

Q1) Please explain how the stack is used for subroutine call and return.

Ans: A stack is used to store data when the most recently used data will also be the first needed. For that reason, stacks are also known as LIFO, for last-in, first-out, structures. A call stack uses this data structure to store the return addresses and arguments of subroutine calls. Stacks are an excellent method for storing the return addresses and arguments from subroutine calls. Suppose the return address were stored in a fixed location, if the routine is called a second time, from within itself, the original returning address is lost and replaced by the new return address. The program is stuck in an infinite loop between. If, the return address is stored on a stack. when the routine is again called, the original address is simply pushed down the stack, below the most recent address. This is exactly what we want: we always return from the last called subroutine to the one just previous. When a subroutine is called, program control is transferred from the main program to the subroutine. When the subroutine finishes executing, control is returned to the main program. The stack provides the means of connecting the subroutines to the main program.

Q2) Explain a computer's register-level architecture, including,

1. CPU-memory interface
2. Special-use registers
3. Addressing modes

Ans: A register is a single, permanent storage location within the CPU used for a particular, defined purpose. A register is used to hold a binary value temporarily for storage, for manipulation, and/or for simple calculations and each register is wired within the CPU to perform its specific role. The register’s size, the way it is wired, and even the operations that take place in the register reflect the specific function that the register performs in the computer.

a) CPU-Memory Interface: Two registers, the memory address register and the memory data register, act as an interface between the CPU and memory. The memory address register holds the address in the memory that is to be “opened” for data. The MAR is connected to a decoder that interprets the address and activates a single address line into the memory. There is a separate address line for each row of cells in the memory; thus, if there are n bits of addressing, there will be 2n address lines. The memory data register is designed such that it is effectively connected to every cell in the memory unit. Each bit of the MDR is connected in a column to the corresponding bit of every location in memory. The interaction between the CPU and the memory registers takes place as follows: to retrieve or store data at a particular memory location, the CPU copies an address from some register in the CPU to the memory address register. Note that addresses are always moved to the MAR; there would never be a reason for an address transfer from the MAR to another register within the CPU, since the CPU controls memory transfers and is obviously aware of the memory address being used. At the same time that the MAR is loaded, the CPU sends a message to the memory unit indicating whether the memory transfer is a retrieval from memory or a store to memory. This message is sent by setting the read/write line appropriately. At the appropriate instant, the CPU momentarily turns on the switch that connects the MDR with the register by using the activation line, and the transfer takes place between memory and the MDR. The MDR is a two-way register. When the instruction being executed is to store data, the data will be transferred from another register in the CPU to the MDR, and from there it will be transferred into memory. The original data at that location will be destroyed, replaced by the new data from the MDR. Conversely, when the instruction is to load data from memory, the data is transferred from memory to the MDR, and it will subsequently be transferred to the appropriate register in the CPU. In this case, the memory data are left intact, but the previous data value in the MDR is replaced by the new data from memory.

b) Special Purpose Registers: A Special Function Register (or Special Purpose Register, or simply Special Register) is a register within a microprocessor, which controls or monitors various aspects of the microprocessor's function. Depending on the processor architecture, this can include, but is not limited to: I/O and peripheral control (such as serial ports or general-purpose IOs), timers, stack pointers, program counter, stack limit, subroutine return address, 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.

c)Addressing modes: Addressing modes are an aspect of the instruction set architecture in most central processing unit (CPU) designs. The various addressing modes that are defined in a given instruction set architecture define how the machine language instructions in that architecture identify the operand(s) of each instruction. An addressing mode specifies how to calculate the effective memory address of an operand by using information held in registers and/or constants contained within a machine instruction or elsewhere. Different computer architectures vary greatly as to the number of addressing modes they provide in hardware. Most RISC architectures have only about five simple addressing modes, while CISC architectures such as the DEC VAX have over a dozen addressing modes, some of which are quite complicated. The IBM System/360 architecture had only three addressing modes; a few more have been added for the System/390. Even on a computer with many addressing modes, measurements of actual programs, indicate that the simple addressing modes account for some 90% or more of all addressing modes used.

**References**

1) Englander, I. (2014). Chapter 7/ 7.6 Stack Instructions. In The architecture of computer hardware and systems software: An information technology approach (pp. 220–221). essay, Wiley.

2) Englander, I. (2014). Chapter 7/ 7.3 The Memory Unit. In The architecture of computer hardware and systems software: An information technology approach (pp. 200–204). essay, Wiley.

3) Wikimedia Foundation. (2021, April 12). Special function register. Wikipedia. Retrieved October 26, 2021, from https://en.wikipedia.org/wiki/Special\_function\_register.

4) Wikimedia Foundation. (2021, August 3). Addressing mode. Wikipedia. Retrieved October 26, 2021, from https://en.wikipedia.org/wiki/Addressing\_mode.