CIS-350
INFRASTRUCTURE TECHNOLOGIES

The CPU and Memory - Chapter 7

Read chapter 7 (Sections 7.0-7.5), pp. 199-218 to the extent it was covered in class and the lecture notes below. Sections 7.6-7.8 on pp. 218-235 will be covered very superficially.

Please also look at the chapter 7 slides posted on BB. They are excellent.

The chapter discusses:

- Similarities/analogies between the LMC and the real computer
- The concept of registers
- Structure and operation of the CPU and memory
- How instructions are executed in the real computer: fetch and execute cycles
- Buses
- Instruction word formats & classification of instructions

The Components of the CPU

See Figures 7.1 & 7.2

Real computer	LMC	Function
ALU (Arithmetic & Logic Unit)	Calculator	Performs arithmetic operations: (+, -, *, /) and logic comparisons.
Control Unit	Little Man	Fetches instructions from memory and monitors their execution.
Memory	Mailboxes	Stores programs & data. Each memory location has the address and the contents.
I/O interface	In-basket & Out-basket	Handles data as it passes between the CPU and various I/O devices.
Program Counter or Instruction Counter	Instruction location counter	Contains the address of the current/next instruction to execute.

Before we cover the machine cycle in detail, we need to discuss the concept of a register and memory operation.

The Concept of Registers

Register

- A single, permanent storage location within the CPU which is used for a particular, single goal or many different purposes
  - May hold
    - values temporarily for storage and/or calculation

- the address of the instruction
- the instruction to be executed

- Is several times faster than memory locations (instant access vs. 10-20 nanoseconds)
- Does not have the address but may be identified by numbers or letters: register #1, register #2, etc.; register K, register L, etc.

Types of registers:

Flag Registers:	indicate special conditions like (an arithmetic carry or overflow), size - 1 bit
Status Register	combines several flag registers
Accumulator: (ACCA)	several in the CPU; hold the data that are used in arithmetic operations as well as the results
Program/Instruction Counter (PC or IC):	holds the address of the current/next instruction to execute
Instruction Register: (IR)	holds the actual instruction being executed
Memory Address Register (MAR):	holds the address of a memory location
Memory Data Register (MDR):	holds a data value that is being written to or read from the memory location currently addressed by the MAR

All the above registers, except the flag registers (1 bit) used to be 32-bit registers. Now registers in many computers are 64-bit or even 128-bit long. There is no direct reference in the LMC to flag registers, status registers, IR, MAR, and MDR. The two registers used in the LMC are Accumulator (Calculator) and Program Counter (Instruction location counter).

Discuss briefly the registers shown in Fig. 7.3 - IBM zSeries - The one we have here at UofL.

Typical operations involving registers:

LDA
ADD
SUB
CLR
INV
ADD-1
SUB-1
SHIFT-R
SHIFT-L

ADD-1 is a very important operation

- to look at the next instruction in the program counter (PC): PC+1 $\rightarrow$ PC

- to count for loops

```
int k=1;
while (k <= 4) {
        print k;
        k=k+1;
}
```

- to index through arrays in programs

```
int x[5]={7, 8, 9, 0, 2}, i=0;
while (i <=4) {
        if (x[i] == 0)
                break;
        i++;
}
```

The Memory Unit

The operation of memory

- built of one-bit cells
- cells are grouped to form bytes
- each group represents the data cells for a single memory address

IBM computers are byte-addressed machines, meaning that each byte has a unique address.

The Memory Address Register (MAR) & Memory Data Register (MDR) act as an interface between CPU and memory.

MAR - holds the address of memory location which is to be "opened" for data R/W. The MAR is connected to a decoder that interprets the address and activates a single address line into the memory. If there are *n* bits addressing, there will be 2^n address lines. MAR is a 1-way register, which means that (1) the address of the instruction is copied from the program/instruction counter to MAR or (2) the address of data is extracted from the instruction register and copied to MAR. You will understand the last sentence when we work an example.

MDR - holds data that is to be R/W from/to memory. Each bit is connected to the corresponding bit of every location in memory. MDR is a 2-way register, which means that depending on the instruction its contents is written into memory (STO) or read from memory (LDA).

Look at Figures 7.4-7.6 in a general way.

Memory capacity is determined by two factors:

Primary factor:

> The # of bits in the MAR determines how many different address locations can be decoded; more importantly, how many physical memory locations can be addressed.
>
> $M=2^k$ *k* - register width (in bits)
> M - the # of possible memory locations

Secondary factor:

> The # of bits in the address field of the instruction, which establishes how many memory locations can be directly addressed from that instruction.

LMC		Real computer	
one digit	two digits	6 or 8 bits	8, 12 or 24 bits
opcode	operand	opcode	operand

This is the less important factor because the address field (operand) can be extended by utilizing, for example, another register from the CPU.

The address part of the instruction is usually kept short (12 bits) to keep the length of instructions small and save storage space.

Old PCs: MAR - 20 bits --> 2^{20} = 1MB; MAR - 24 bits --> 2^{24} = 16MB
Today's PCs: MAR - 32 bits --> 2^{32} = 4GB, MAR - 64 bits → 2^{64}=16 billion gigabytes

This is the addressing capacity - the size of MAR. It does not mean that the address part/field of the instruction is also 32 bits or 64 bits. The MAR always contains the physical/real/absolute address, whereas the address field of the instruction contains the logical/relative address.

The MDR is designed to retrieve/store:
- data
- instructions

in a single fetch operation.

The size of the MDR (usually 4 or 8 bytes) determines the size of the word transferred between the CPU, memory, and MDR.

In the interest of speed, it is desired to retrieve/store several consecutive bytes (4 or 8) during a single R/W operation.

Reasons:
- some instructions are: 2-byte long (R <--> R)
  4-byte long (M <--> R)
  6-byte long (M <--> M)
- double words which enable more precision are stored in 8 bytes (64 bits) Ex. in C++/Java: double pi=3.1415987277637...

We would like to read the entire instruction or the double word of data, not a portion of it.

Memory Characteristics and Implementations

Main memory:
- built of chips, integrated circuits that can be in 2 states: 0 or 1
- limited capacity
- volatile

Technical considerations:
- cost – generally inexpensive
- the speed of memory access
- the total amount of memory that can be addressed
- the data width
- bit density

RAM - Random Access Memory
- typical size: 1GB, 2GB, 4GB on the PC
- can R/W (W operation erases the previous contents of a memory location)
- available to the user/programmer (accept some protected areas where the OS resides)

- the OS allocates memory to programs and data

SRAM (Static RAM)

more expensive
faster access
no need to refresh
requires more chips (less bits of storage in a single chip)
used in fast/cache memories

DRAM (Dynamic RAM)

has the opposite features to SRAM, used in conventional memory

Memory hierarchy

	CPU registers (technically, not memory)	Cache memory (SRAM)	RAM (DRAM)	Flash memory	Disk drive
Storage capacity	1-128 bits	512KB-1MB	1-4GB	8-16GB	200-300GB
Access time	instant	1-10ns ($1ns=10^{-9}s$)	10-50ns	120μs ($1\mu s=10^{-6}s$)	10-50ms ($1ms=10^{-3}s$)

The CPU looks for instructions and data in cache memory first - fast access

Flash Memory

- Relatively inexpensive, portable, and non-volatile storage
- Slow access time and limited number of rewrites compared to RAM makes it an unsuitable replacement for RAM
- Due to faster access time than hard disk viewed as a potential replacement for hard disk, but the cost and technology are still prohibitive

ROM - Read Only Memory
- may store some OS routines, I/O drivers, boot routines
- not available to the user/programmer
- non-volatile

Some modern ROMs allow changing its contents using special circuitry. For example,

In engineering applications:
EEPROM - Erasable Electrically Programmable ROM is used

The Machine Cycle (Instruction Cycle)

Most of the machine cycle consists of copying data from one register to another.

The "from" register → The "to" register

The contents of the "to" register is always affected.

Machine cycle consists of the Fetch phase and the Execute phase.

Fetch phase: retrieving the instruction from memory

step 1: PC --> MAR	The contents of the PC is copied to the MAR. PC holds the address of an instruction to be executed. The instruction is copied from the memory location specified by the MAR to the MDR.
step 2: MDR --> IR	The instruction is copied from the MDR to the IR. The IR will interpret and hold the instruction through the cycle.

Execute phase (for LOAD - LDA):

step 3: IR [address] --> MAR	The address of data to act upon is copied from the IR to the MAR. In other words, only the address field/part of the instruction is copied. The data residing at the memory location specified by the MAR is copied to the MDR.
step 4: **MDR --> ACCA**	The contents of MDR is sent to ACCA.
step 5: PC + 1 --> PC	The CPU increments the PC. PC points to the next instruction now.

The exercise to be worked in class on the white board. The machine cycle for the LDA instruction (pictorially).

The PC could also be incremented earlier in the machine cycle, for example, right after the instruction is brought to the IR in step 2.

The machine cycle for STORE - STO

PC --> MAR
MDR --> IR

IR [address] --> MAR
ACCA --> MDR
PC+1 --> PC

The machine cycle for ADD

PC --> MAR
MDR --> IR

IR [address] --> MAR
ACCA + MDR --> ACCA
PC+1 --> PC

You can see that step 4 (in bold) only differs.

Exercise to be worked in class on the white board. The machine cycle for the STO (or ADD) instruction (pictorially).

Buses

Bus lines

Components of a computer system are physically linked by bus lines.
A bus is a set of parallel or serial wires, or optical conductors that can carry bits.

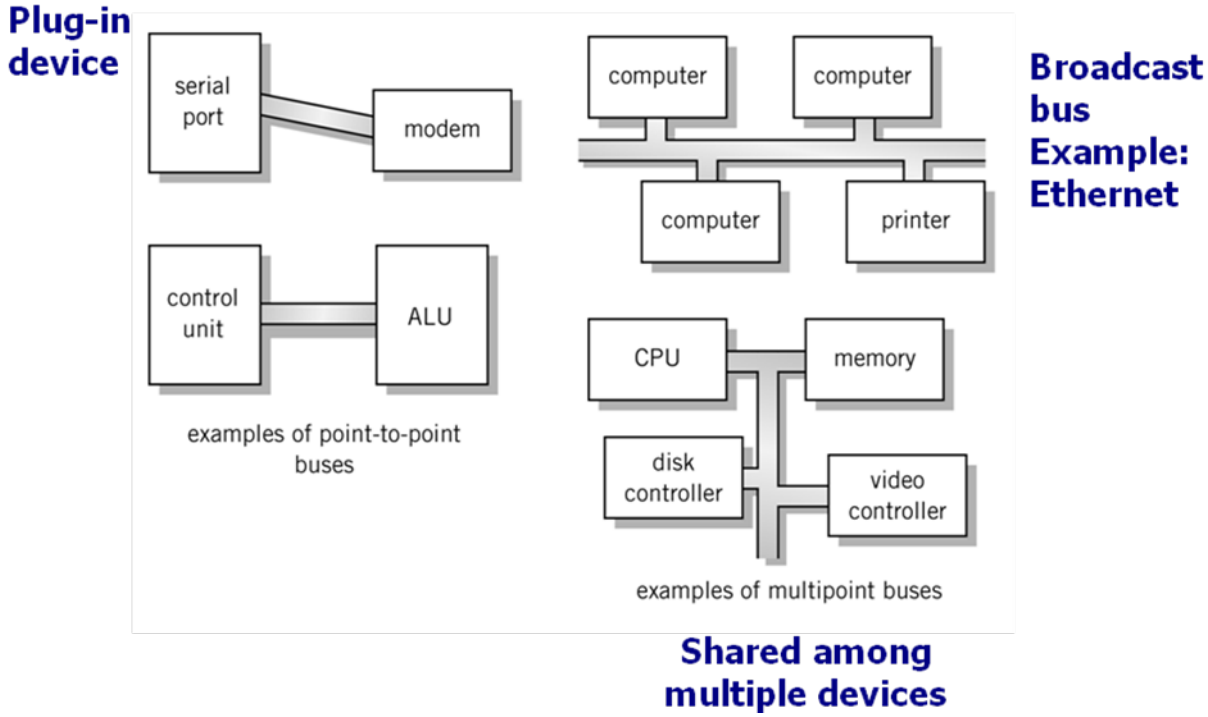
The bus can:
- transmit power
- carry instructions
  - data
  - addresses
  - commands
  - control signals

Typically, elements in the computer are connected by multiple buses through which all information is transmitted. Registers within the CPU are connected by very tiny buses imprinted on integrated circuits. Similarly, CPU is connected with memory by buses or I/O devices peripherals) are connected to the computer by buses (cables) plugged to the respective ports.

Buses are characterized by:
- throughput (data transfer rate measured in bits/sec)
- the data width (wider bus - more data transferred)
- the number and types of lines on the bus: a line(s) can carry addresses, data or control signals, or all of them - lines on the bus can be shared
- the distance between two end points
- type of control required
- # of pins on the connector
- voltage used

Bus Categorizations:

- Parallel vs. serial buses
- Direction of transmission
  - Simplex – unidirectional
  - Half duplex – bidirectional, one direction at a time
  - Full duplex – bidirectional simultaneously
- Method of interconnection
  - Point-to-point – single source to single destination
    - Cables – point-to-point buses that connect to an external device
    - Do not require addresses of the destination device
  - Multipoint bus – also broadcast bus or multidrop bus
    - Connect multiple points to one another
    - In Ethernet network, signals sent by a particular computer on a network are received by every other computer connected to the network
    - May require address of the destination device



Parallel vs. Serial Buses

- Parallel
  - High throughput because all bits of a word are transmitted simultaneously
  - Expensive and require a lot of space
  - Subject to radio-generated electrical interference which limits their speed and length
  - Generally used for short distances such as CPU buses and on computer motherboards
- Serial
  - 1 bit transmitted at a time
  - Single data line pair and a few control lines
  - For many applications, throughput is higher than for parallel because of the lack of electrical interference

Lines on the bus are assigned names to make individual lines easier to identify.
Buses themselves are assigned names: PCI, FSB. Express PCI or PCI is a popular modern bus used in many different types of computers. The interfaces between buses are called bridges.

Signals on the bus can be multiplexed in time: first control signals or addresses are sent followed by the data.

Instructions in the computer

The sections discuss:

- instruction word formats
- instruction word requirements and constraints, and
- classification of instructions

The intention of the sections is neither to overwhelm you with many details to memorize nor to make you programming in assembly language, but to introduce you to the major approaches used in modern computers.

Each computer has its own unique instruction set that depends on the overall design of the computer (Ex. the # of accumulators).

Instruction words vary in format, in size, and in the number of arguments. The various instructions and addressing modes (will not be discussed in this course) add flexibility and convenience for the programmer.

Instruction Word Formats

LMC
10 instructions in its instruction set
- LDA
- STO
- Branch (BRZ, BR, BRP)
- IN
- OUT
- ADD
- SUB
- HLT

Real computer
- 50-150 instructions in its instruction set
- similar to the LMC's instructions
- instructions are more sophisticated, more efficient, and enable easier programming

RISC (Reduced Instruction Set Computer)
- 15-20 instructions
- the speed of their execution is optimized (division of integer #s)
- engineering applications: digital signal processing, computer vision, game applications

LMC

1 digit	2 digits
opcode	operand
tells what to do	address of memory locations on which the opcode acts

STO 90 (390) – 3 or STO is the opcode and 90 is the memory address where the data resides

ADD 95 (195) – 1 or ADD is the opcode and 95 is the memory address where the data resides

IN (901) – 9 is the opcode for Read and 01 is the operand that indicates an address of the input device

BRP 02 (802) – 8 or BRP is the opcode and 02 is the address of the next instruction to be executed

BR 90 (690) – 6 or BR is the opcode and 90 is the address of the next instruction to be executed

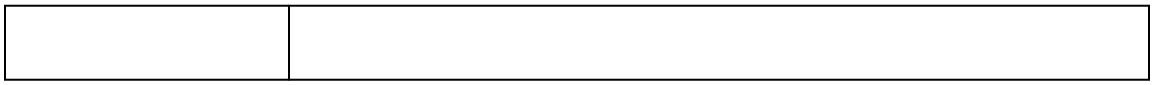
HLT (000) – 0 or HLT is the opcode and operand 00 is not meaningful

All instructions above have:
- one operand (that usually indicates the address)
- no operand or operand not meaningful

Real computer

Instructions may have:
- no operands
- one operand or more operands

32-bit instruction word size



8-bit opcode (2^8=256 unique instructions)

24-bit operand (address field) may indicate/include:
- memory addresses (2^{24} = about 16 million addresses)
- register #s
- the # of bits to shift
- addressing modes

Implicit destination/source

In general, computer instructions that manipulate data require the specification of at least two locations for the data: one or more source locations and one destination location. These locations may be expressed explicitly, as address fields in the instruction word, or implicitly, as part of the definition of the instruction itself.

In the LMC computer, for data movement instructions the destination was typically either:
- a memory location (STO); the source is implicit (the accumulator)
- an accumulator (LDA, ADD, SUB); the destination is implicit (the accumulator); the source of the data is the explicit memory location

If the CPU has one accumulator register, there is no need to name it.

Ex. LDA 50 (one operand)

The accumulator is an implicit destination, not mentioned in the instruction itself.

Ex. STO 50

The accumulator is an implicit source of data.

Ex. SHIFT-R 2 (2 indicates how many bits to shift)

The accumulator is an implicit source and destination. The instruction acts on the accumulator only.

Before:	A	00001010
After:	A	00000010

Explicit source/destination

In case of two or more accumulators, you need to give the accumulator #.
Ex. LDA 3, 50

Two operands:
- 3 is the accumulator # - destination
- 50 is the address of memory location - source

The instruction word size requirements and constraints
- CPU dependent
- may vary

Sun Sparc processor:	4 bytes
Pentium processor:	1-15 bytes, most are 1-2 byte instructions
IBM zSeries:	2 bytes (R ↔ R), 4 bytes (R ↔ M), 6 bytes (M ↔ M)

Classifications/categories of instructions:

1. Privileged instructions
- executed by the OS routines to
  - provide security
  - control access to memory or access to peripheral devices
- the application program does not execute them

Examples:

SVC 33 (IBM - increase storage)
HLT - to stop

A programmer can issue a privileged instruction within the application program, but program control is surrendered to the OS; the application program does not control storage allocation, for example.

2. Non-privileged instructions

- executed by the application program
- four categories:
  - data movement
  - arithmetic
  - program control
  - stack

A. Data movement instructions

- very frequently used
  - R ↔ R v. fast, no memory access, registers are within the CPU
  - M ↔ R, slower, one memory access
  - M ↔ M slowest, two memory access
- use different addressing modes (will not discuss it)

Ex. LOAD 200000

load to ACCA the contents of a 2-byte or a 4-byte word starting at address 200000

B. Arithmetic instructions

ADD
SUB
MUL (the name is made up)
DIV (the name is made up)

- on integer #s
- on floating point #s

Note that that the CPU has separate registers for performing the arithmetic on integers and real numbers (Fig. 7.3, p. 204)

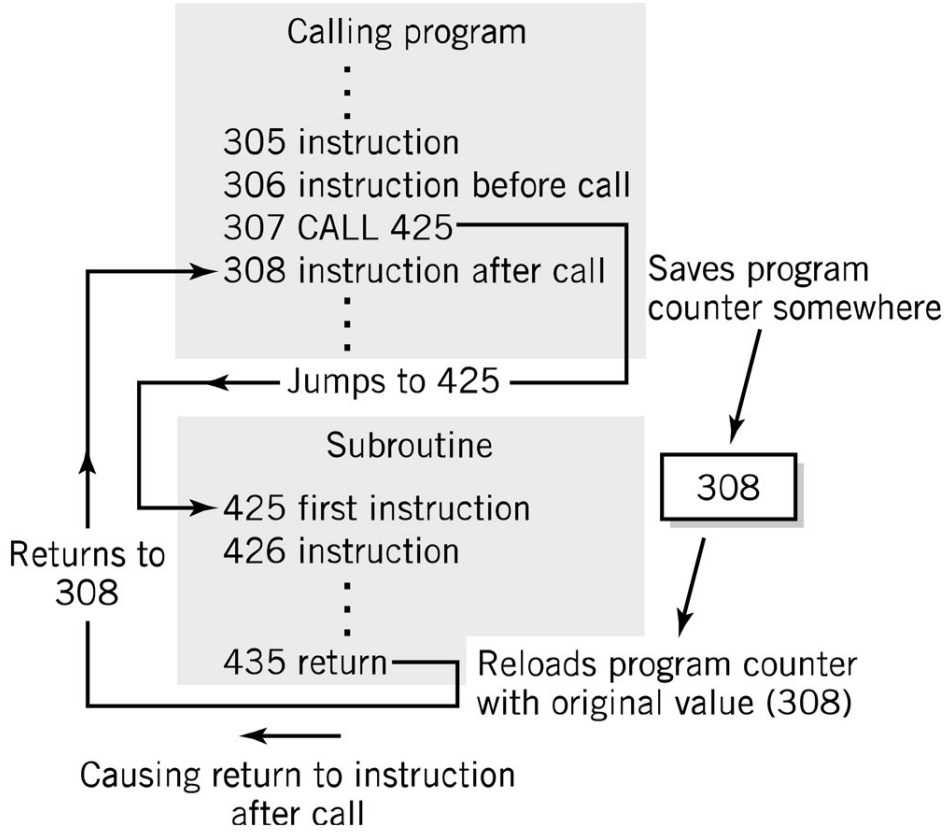
- IBM/z series - 16 registers for floating point arithmetic, 64 bits each
  - 16 registers for integer arithmetic, 64 bits each
  - two 64-bit registers can be combined to form a 128-bit register

Math coprocessor is installed to speed up arithmetic operations.

C. Program control instructions include:

- conditional (BRP, BRZ) and unconditional branches (BR)
- subroutine/function calls (CALL)
- return instructions (RETURN)

Ex.



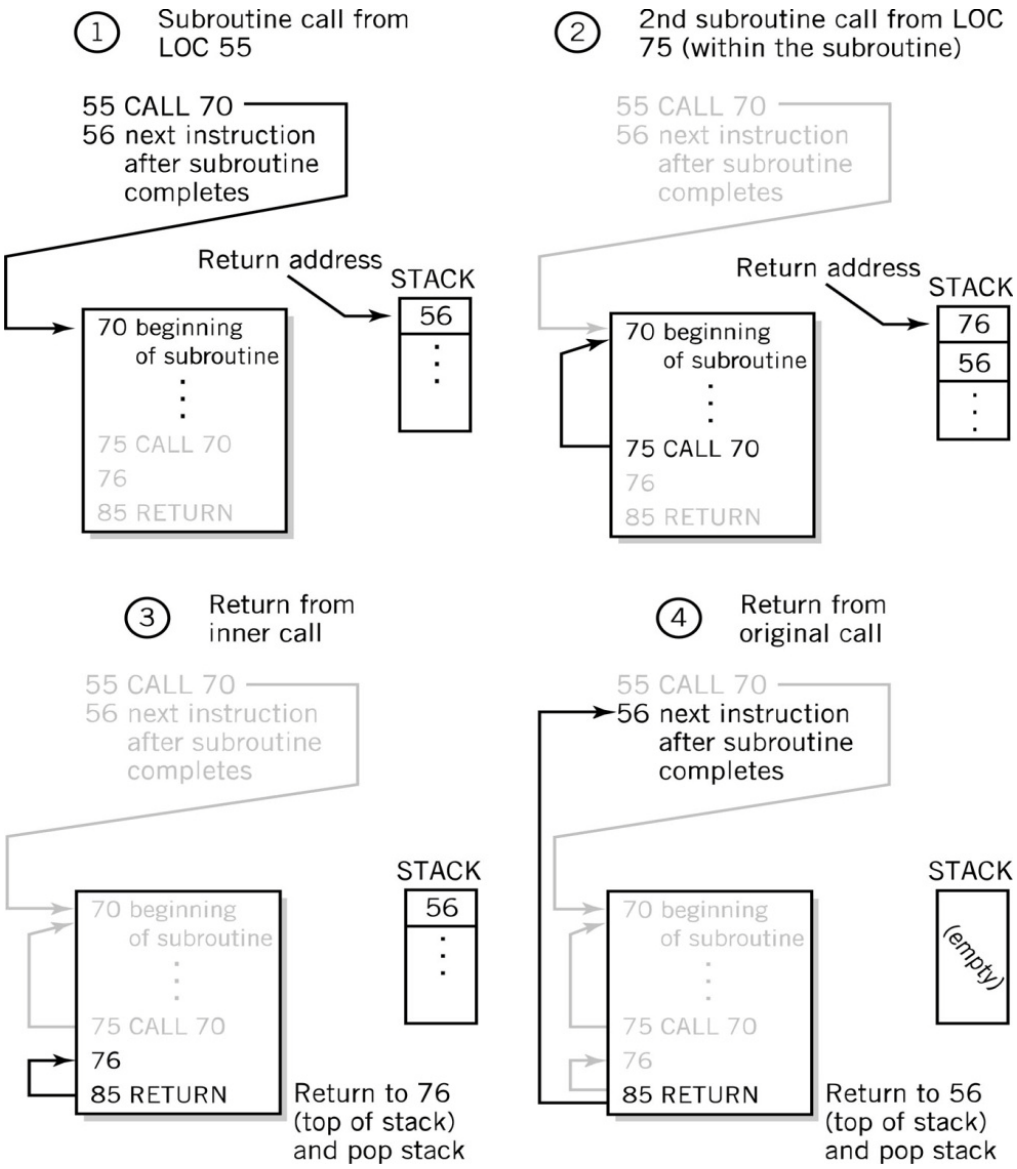
D. Stack instructions

Stack

- data storage structure, v. important in programming
- operates in a LIFO (last-in, first-out) manner
- items are pushed (written) on the top of the stack, or popped (read from) the top of the stack
- the OS maintains the stack pointer (SP) which always points to the top of the stack - the most recent entry
- recursive calls (a subroutine can call itself) are implemented using a stack; the return addresses are preserved on a stack

See Fig. 7.13, p. 225

Ex.



Please also look at the chapter 7 slides posted on BB. They are excellent.