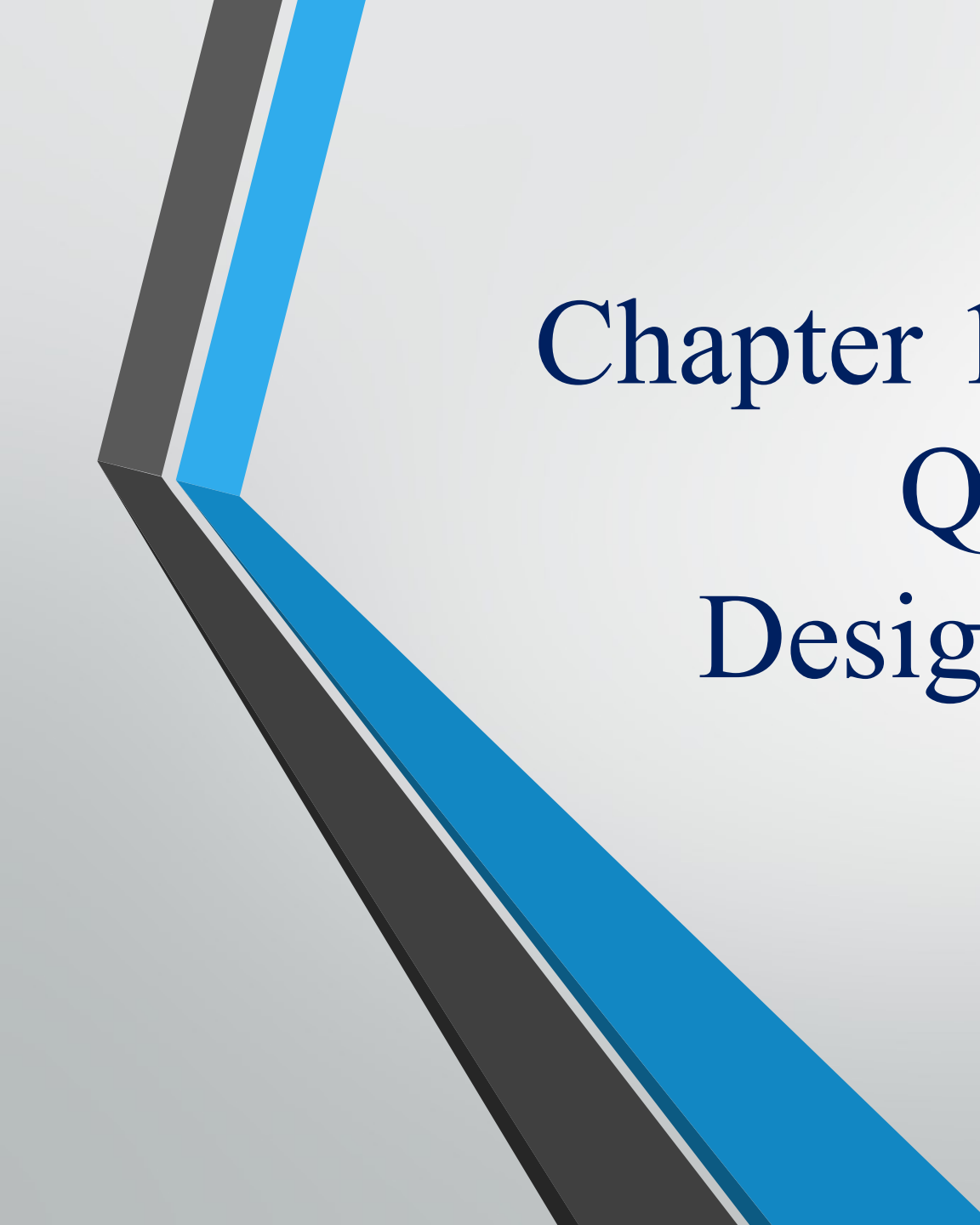





Lecture 3

Dr: Alshaimaa Mostafa Mohammed



Chapter 1: Fundamentals of Quantitative Design and Analysis



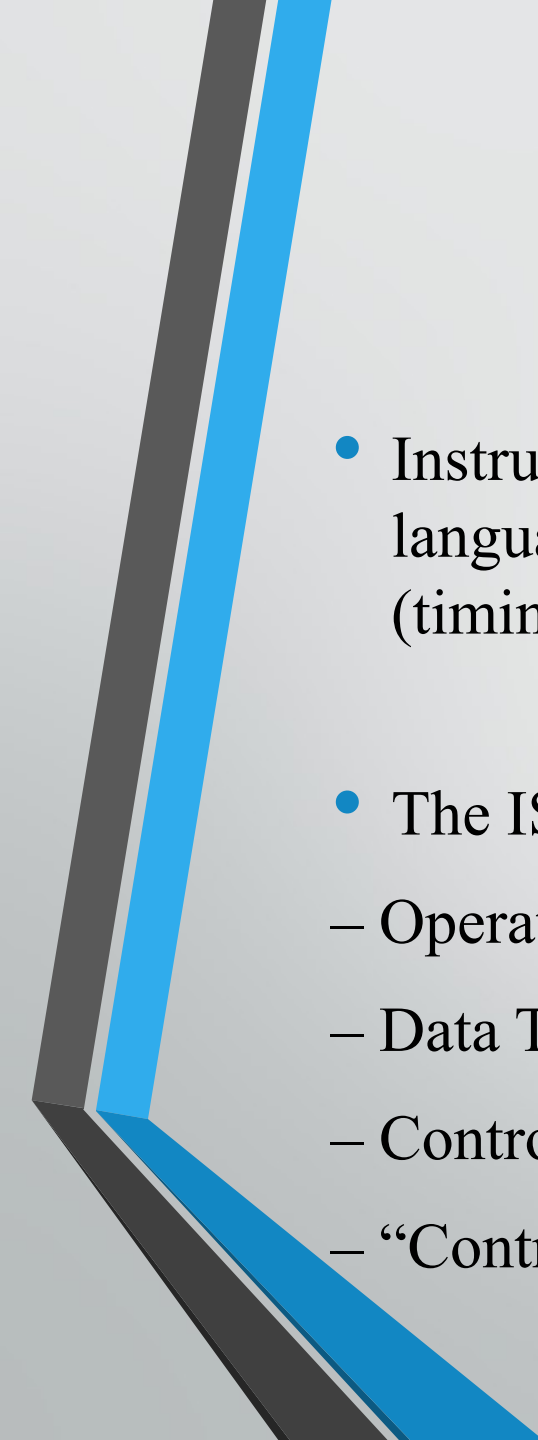
William Stallings

Computer Organization and Architecture

Chapter 12: Instruction Sets: Characteristics
and Functions

Instruction Set Architecture

- *Instruction Set Architecture* (ISA) to refer to the actual programmer visible instruction set.
- The ISA serves as the boundary between the software and hardware.

- 
- Instruction Set Architecture is the structure of a computer that a machine language programmer (or a compiler) must understand to write a correct (timing independent) program for that machine.”
 - IBM, Introducing the IBM 360 (1964)
 - The ISA defines:
 - Operations that the processor can execute
 - Data Transfer mechanisms + how to access data
 - Control Mechanisms (branch, jump, etc)
 - “Contract” between programmer/compiler + HW

What leads to a good/bad ISA?

- Ease of Implementation (Job of Architect/Designer)
 - Does the ISA lead itself to efficient implementations?
- Ease of Programming (Job of Programmer/Compiler)
 - Can the compiler use the ISA effectively?
- Future Compatibility
 - ISAs may last 30+ yrs
 - Special Features, Address range, etc. need to be thought out

Implementation Concerns

- Simple Decoding (fixed length)
- Compactness (variable length)
- Simple Instructions (no load/update)
 - Things that get microcoded these days
 - Deterministic Latencies are key!
 - Instructions with multiple exceptions are difficult
- More/Less registers?
 - Slower register files, decoding, better compilers
- Condition codes/Flags (scheduling!)

What is an Instruction Set?

- The operation of the processor is determined by the instructions it executes, referred to as **machine instructions** or **computer instructions**.
- The collection of different instructions that the processor can execute is referred to as the **processor's instruction set**.
- The complete collection of instructions that are understood by a CPU
- Machine Code
- Binary
- Usually represented by assembly codes

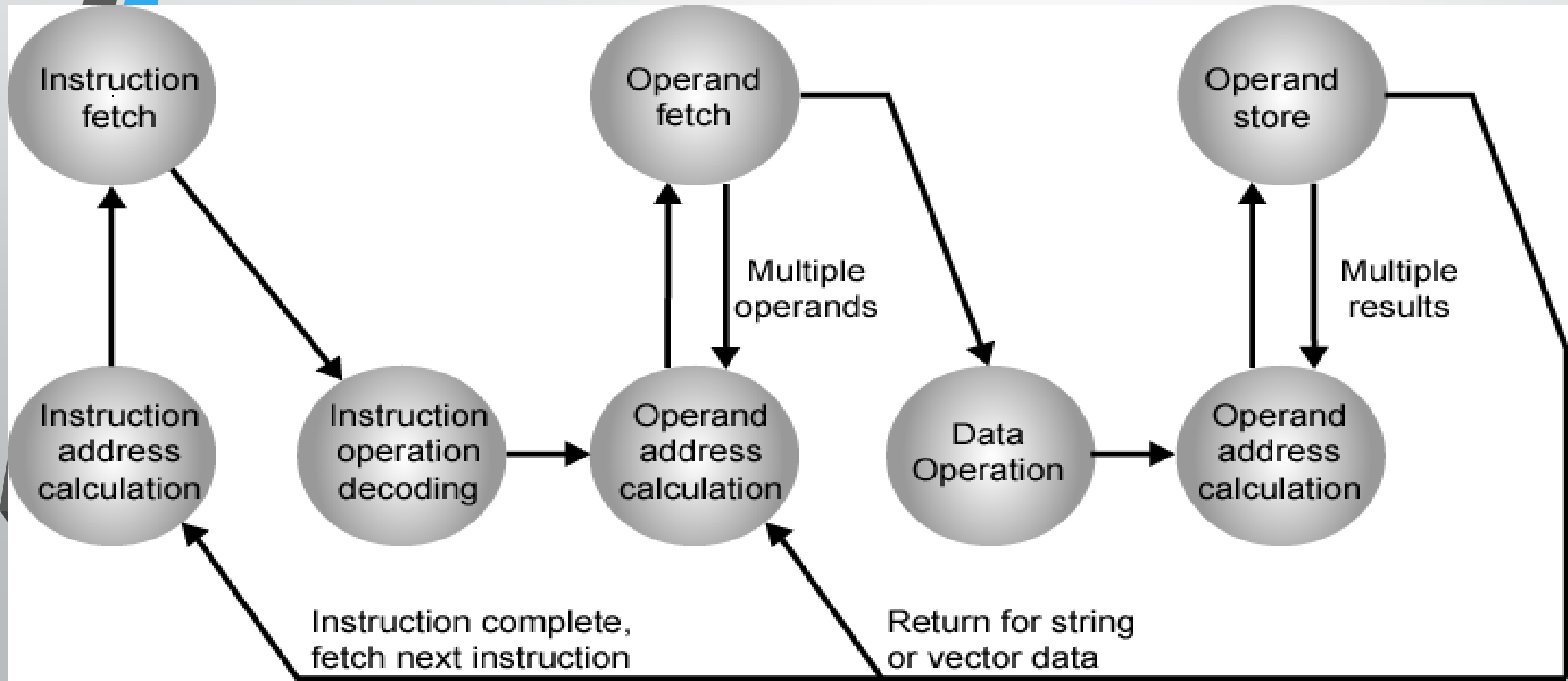
Elements of an Instruction

- Operation code (Op code)
 - Do this
- Source Operand reference
 - To this
- Result Operand reference
 - Put the answer here
- Next Instruction Reference
 - When you have done that, do this...

Areas of Source and result operands

- **Main or virtual memory:** As with next instruction references, the main or virtual memory address must be supplied.
- **Processor register:** With rare exceptions, a processor contains one or more registers that may be referenced by machine instructions. If only one register exists, reference to it may be implicit. If more than one register exists, then each register is assigned a unique name or number, and the instruction must contain the number of the desired register.
- **Immediate:** The value of the operand is contained in a field in the instruction being executed.
- **I/O device:** The instruction must specify the I/O module and device for the operation. If memory- mapped I/O is used, this is just another main or virtual memory address.

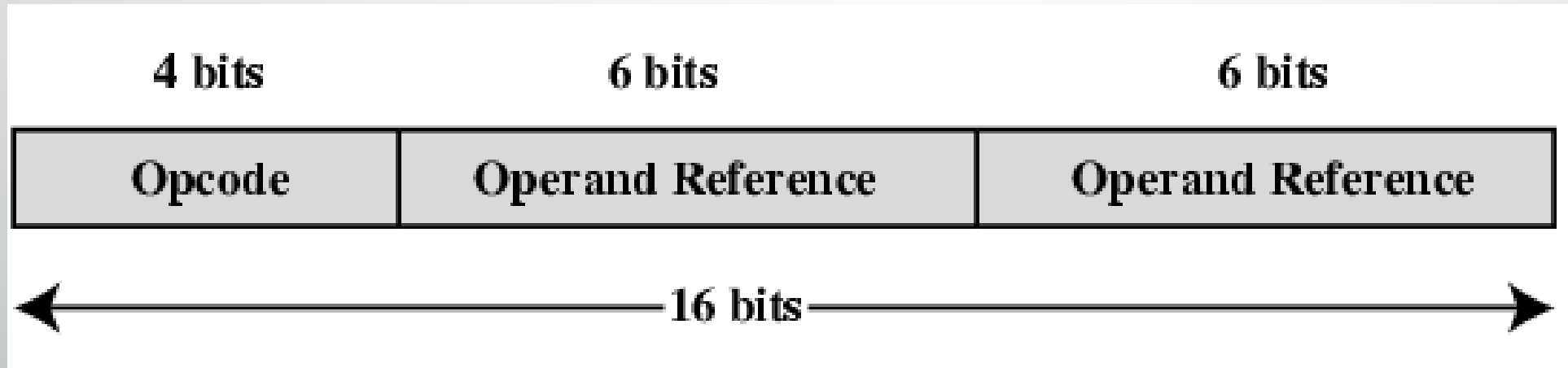
Instruction Cycle State Diagram



Instruction Representation

- In machine code each instruction has a unique bit pattern
- For human consumption (well, programmers anyway) a symbolic representation is used
 - e.g. ADD, SUB, LOAD
- Operands can also be represented in this way
 - ADD A,B

Simple Instruction Format



Instruction Types

- **Data processing:** Arithmetic and logic instructions.
- **Data storage (main memory):** Movement of data into or out of register and or memory locations.
- **Data movement (I/O):** I/O instructions.
- **Program flow control:** Test and branch instructions.

Number of Addresses (a)

- 3 addresses
 - Operand 1, Operand 2, Result
 - $a = b + c;$
 - May be a forth - next instruction (usually implicit)
 - Not common
 - Needs very long words to hold everything

Number of Addresses (b)

- 2 addresses
 - One address doubles as operand and result
 - $a = a + b$
 - Reduces length of instruction
 - Requires some extra work
 - Temporary storage to hold some results

Number of Addresses (c)

- 1 address
 - Implicit second address
 - Usually a register (accumulator)
 - Common on early machines

Design Decisions (1)

- The most important of these fundamental design issues include the following:
- **Operation repertoire:** How many and which operations to provide, and how complex operations should be.
- **Data types:** The various types of data upon which operations are performed.
- **Instruction format:** Instruction length (in bits), number of addresses, size of various fields, and so on.
- **Registers:** Number of processor registers that can be referenced by instructions, and their use.
- **Addressing:** The mode or modes by which the address

Types of Operand

- Machine instructions operate on data. The most important general categories of data are
 - Addresses
 - Numbers
 - Integer/floating point
 - Characters
 - ASCII etc.
 - Logical Data
 - Bits or flags
- (Aside: Is there any difference between numbers and characters? Ask a C programmer!)

x86 Data Types

- 8 bit Byte
- 16 bit word
- 32 bit double word
- 64 bit quad word
- 128 bit double quadword
- Addressing is by 8 bit unit
- Words do not need to align at even-numbered address
- Data accessed across 32 bit bus in units of double word read at addresses divisible by 4
- Little endian

(READ)

ARM Data Types

- 8 (byte), 16 (halfword), 32 (word) bits
- Halfword and word accesses should be word aligned
- Nonaligned access alternatives
 - Default
 - Treated as truncated
 - Bits[1:0] treated as zero for word
 - Bit[0] treated as zero for halfword
 - Load single word instructions rotate right word aligned data transferred by non word-aligned address one, two or three bytes
 - Alignment checking
 - Data abort signal indicates alignment fault for attempting unaligned access
 - Unaligned access
 - Processor uses one or more memory accesses to generate transfer of adjacent bytes transparently to the programmer

ARM Data Types

- Unsigned integer interpretation supported for all types
- Twos-complement signed integer interpretation supported for all types
- Majority of implementations do not provide floating-point hardware
 - Saves power and area
 - Floating-point arithmetic implemented in software
 - Optional floating-point coprocessor
 - Single- and double-precision IEEE 754 floating point data types

Types of Operation

- Data Transfer
- Arithmetic
- Logical
- Conversion
- I/O
- System Control
- Transfer of Control

Data Transfer

- Specify
 - Source
 - Destination
 - Amount of data
- May be different instructions for different movements
 - e.g. IBM 370
- Or one instruction and different addresses
 - e.g. VAX

Arithmetic

- Add, Subtract, Multiply, Divide
- Signed Integer
- Floating point ?
- May include
 - Increment ($a++$)
 - Decrement ($a--$)
 - Negate ($-a$)

Logical

- Bitwise operations
- AND, OR, NOT

Conversion

- Conversion instructions are those that change the format or operate on the format of data. An example is converting from decimal to binary.
- E.g. Binary to Decimal

Input/Output

- May be specific instructions
- May be done using data movement instructions (memory mapped)
- May be done by a separate controller (DMA)

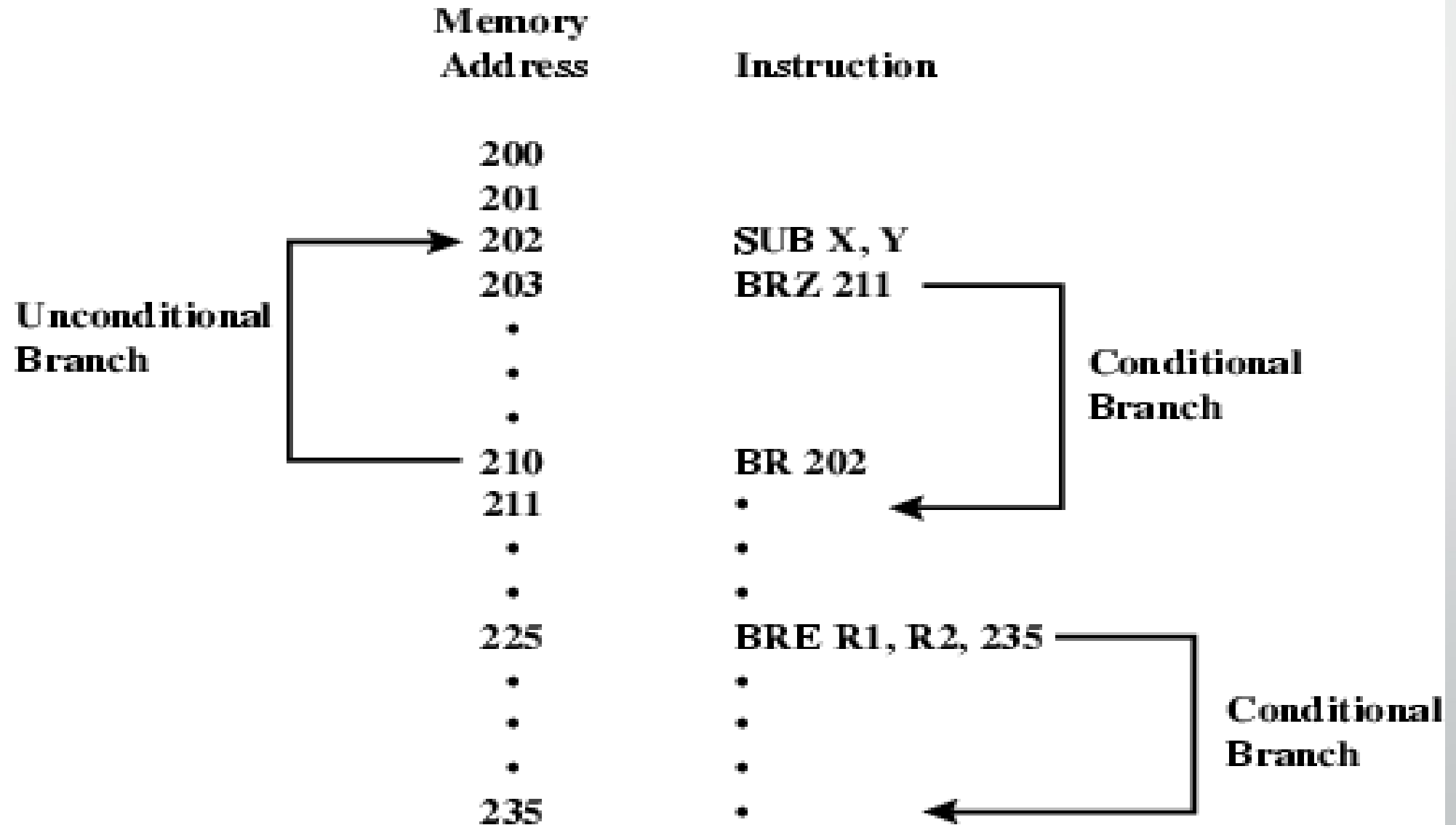
Systems Control

- Privileged instructions
- CPU needs to be in specific state
 - Ring 0 on 80386+
 - Kernel mode
- For operating systems use

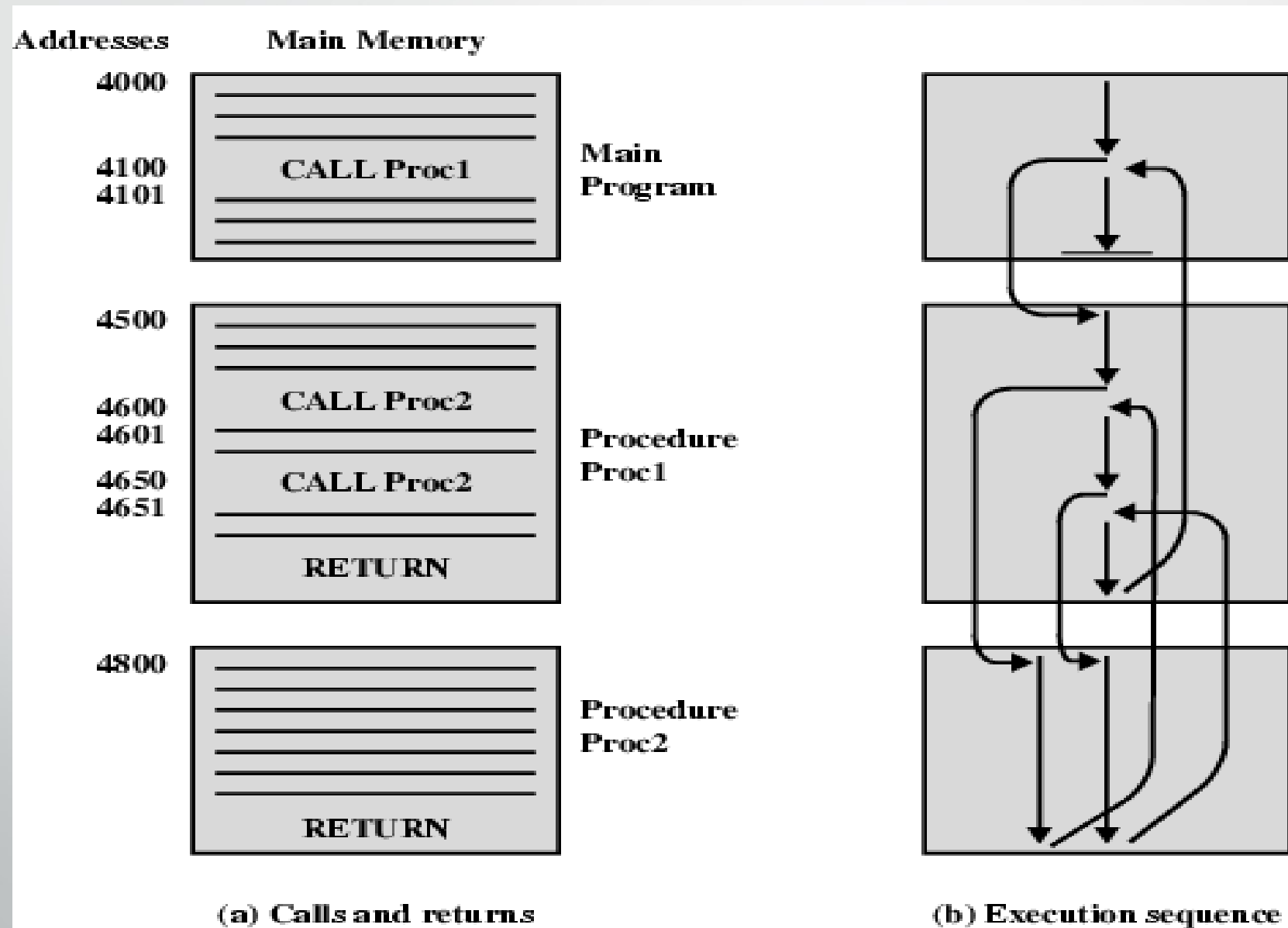
Transfer of Control

- Branch
 - e.g. branch to x if result is zero
- Skip
 - e.g. increment and skip if zero
 - ISZ Register1
 - Branch xxxx
 - ADD A
- Subroutine call
 - c.f. interrupt call

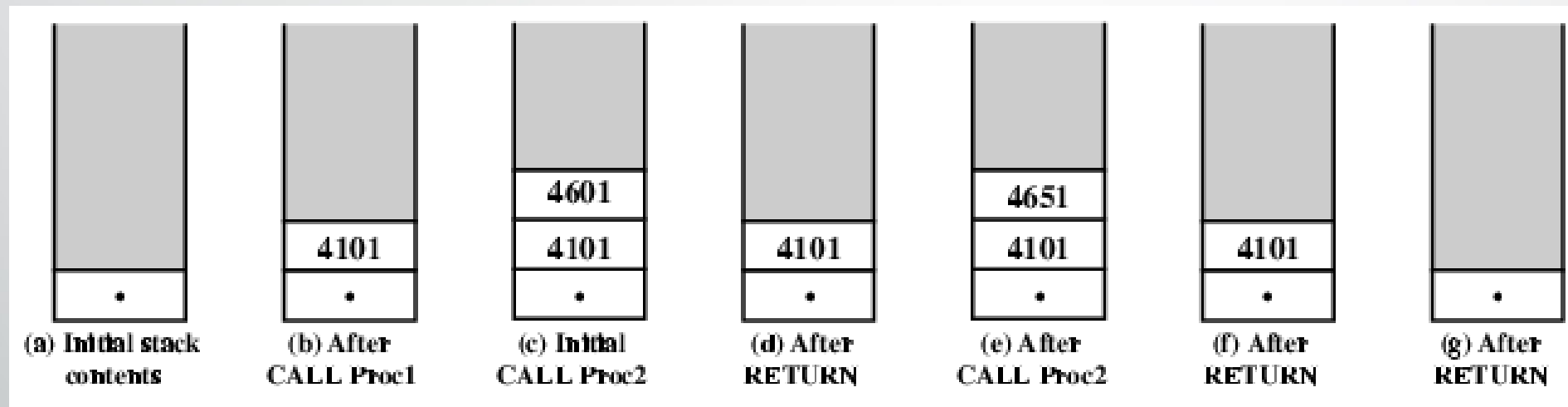
Branch Instruction



Nested Procedure Calls



Use of Stack



Stack Frame Growth Using Sample Procedures P and Q

