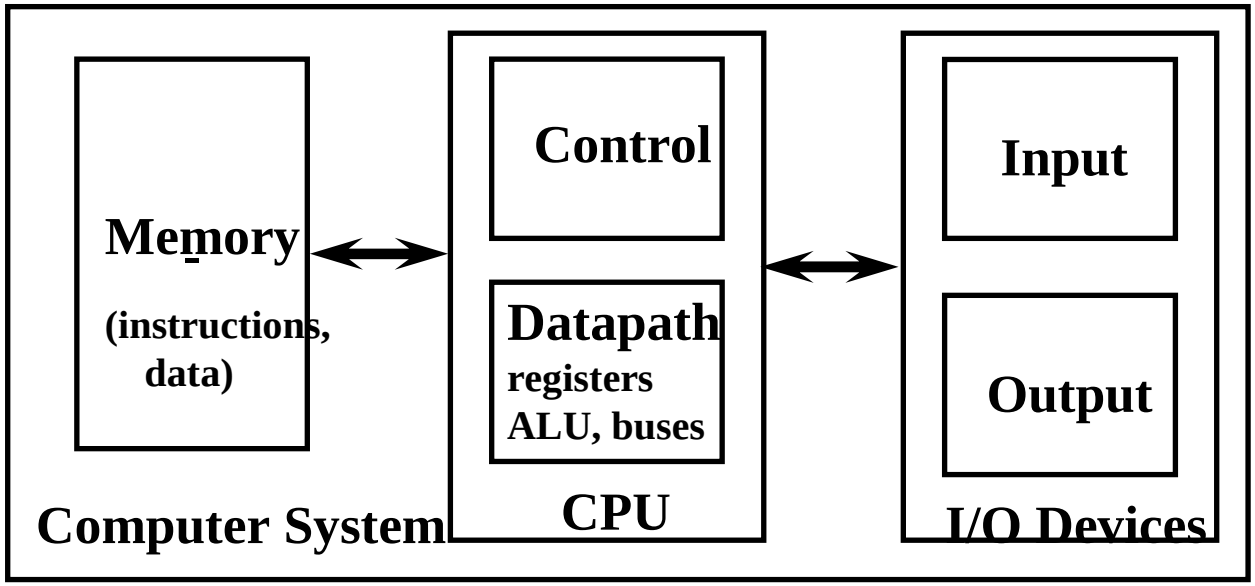
Computer Organization

Instruction Set Characteristics, Instruction Formats, Addressing Modes, RTL & Micro-Operations, CISC, RISC.

Chapters (10 + 11 + Mano Ch.4 + 13)

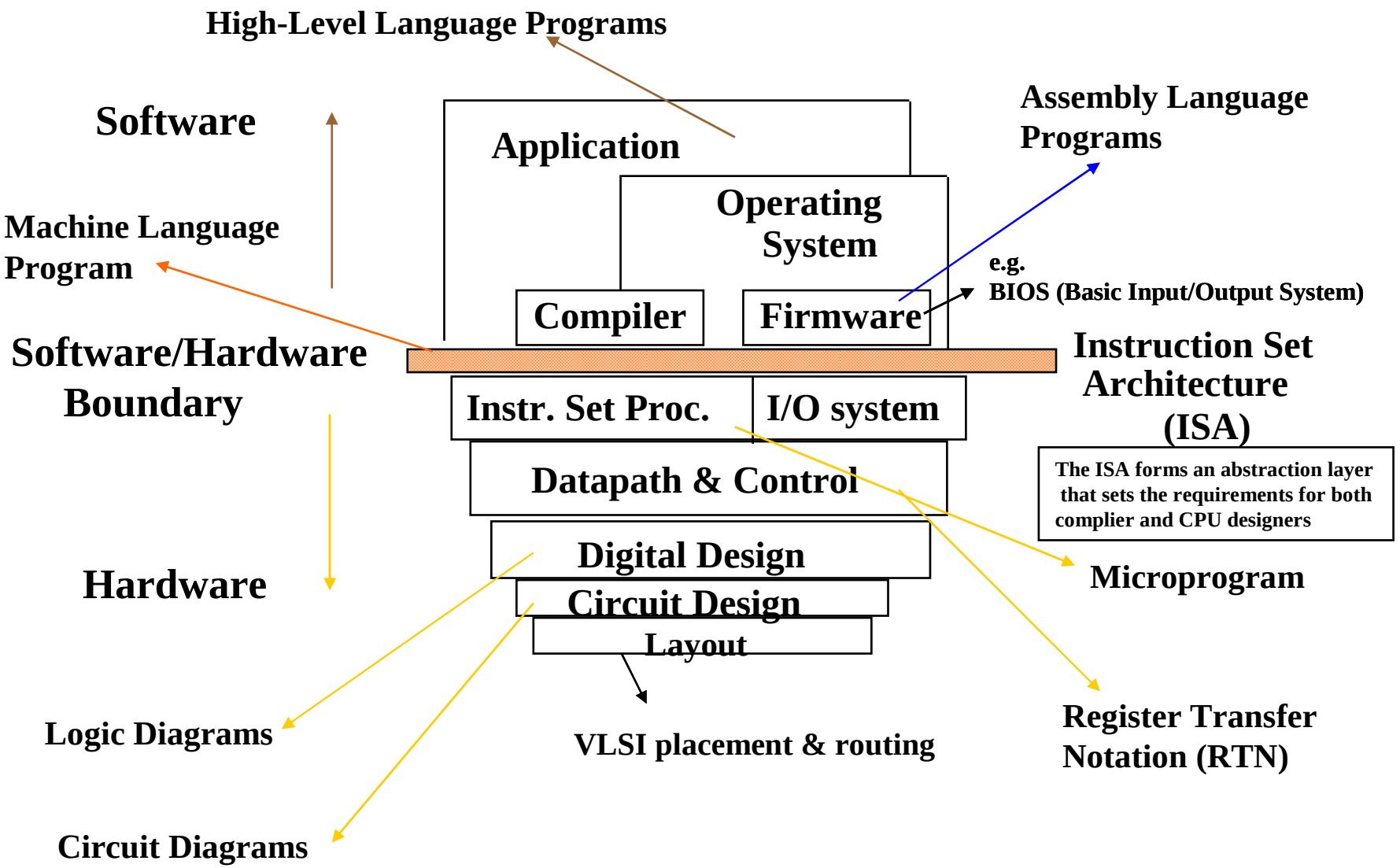
The Von Neumann Computer Model

- Partitioning of the computing engine into components:
  - **Central Processing Unit (CPU):** Control Unit (instruction decode , sequencing of operations), Datapath (registers, arithmetic and logic unit, buses).
  - **Memory:** Instruction and operand storage.
  - **Input/Output (I/O) sub-system:** I/O bus, interfaces, devices.
  - **The stored program concept:** Instructions from an instruction set are fetched from a common memory and executed one at a time

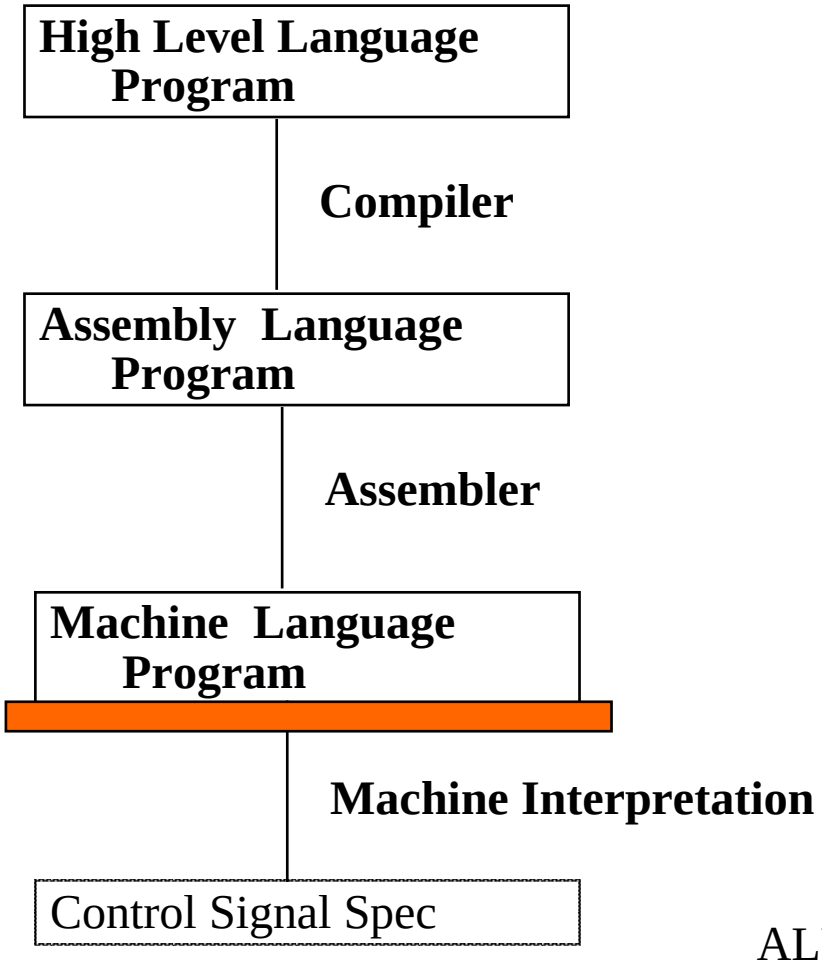


Major CPU Performance Limitation: The Von Neumann computing model implies sequential execution one instruction at a time

Hierarchy of Computer Architecture



How to Speak Computer




```
temp = v[k];
v[k] = v[k+1];
v[k+1] = temp;
```

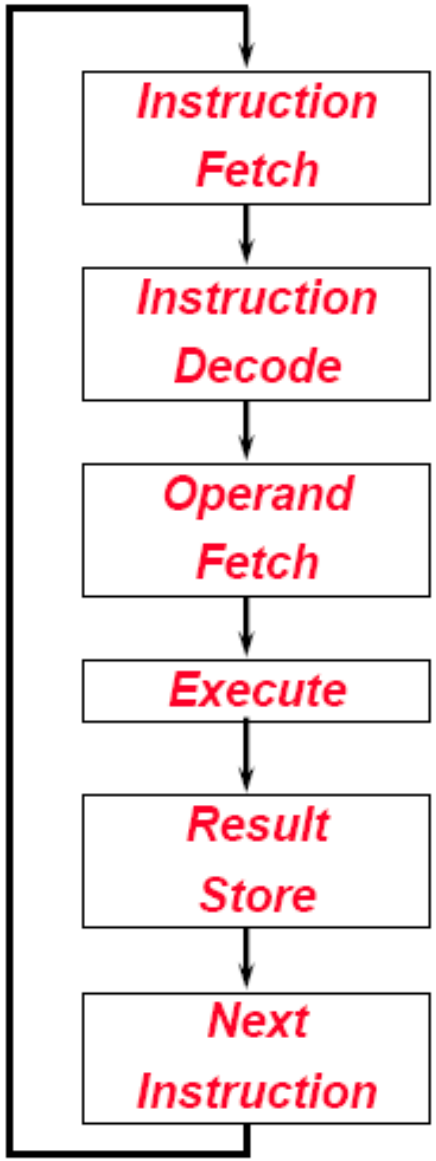
```
lw  $15,   0($2)
lw  $16,   4($2)
sw  $16,   0($2)
sw  $15,   4($2)
```

```
10001100011000100000000000000000
10001100111100100000000000000100
10101100111100100000000000000000
10101100011000100000000000000100
```

ALUOP[0:3] <= InstReg[9:11] & MASK

Need translation from application to physics

Generic CPU Machine Instruction Execution Steps



Obtain instruction from program storage

Determine required actions and instruction size

Locate and obtain operand data

Compute result value or status

Deposit results in storage for later use (if required)

Determine successor instruction

Computing Element Choices

- General Purpose Processors (GPPs): Intended for general purpose computing (desktops, servers, clusters..)
- Application-Specific Processors (ASPs): Processors with ISAs and architectural features tailored towards specific application domains
  - E.g Digital Signal Processors (DSPs), Network Processors (NPs), Media Processors, Graphics Processing Units (GPUs), Vector Processors??? ...
- Co-Processors: A hardware (hardwired) implementation of specific algorithms with limited programming interface (augment GPPs or ASPs)
- Configurable Hardware:
  - Field Programmable Gate Arrays (FPGAs)
  - Configurable array of simple processing elements
- Application Specific Integrated Circuits (ASICs): A custom VLSI hardware solution for a specific computational task
- The choice of one or more depends on a number of factors including:
  - Type and complexity of computational algorithm (general purpose vs. Specialized)
  - Desired level of flexibility/ programmability
  - Performance requirements
  - Development cost/time
  - System cost
  - Power requirements
  - Real-time constrains

Instruction Set Characteristics

Chapter 10

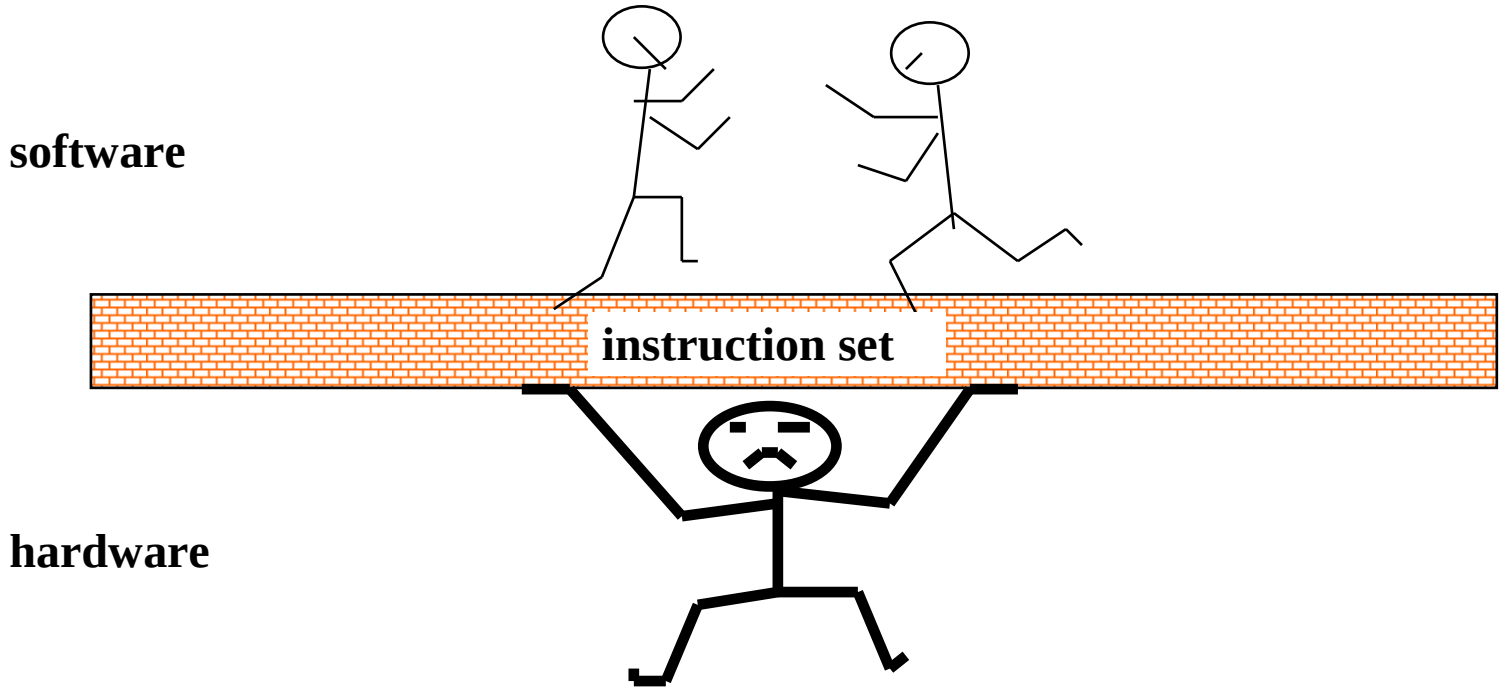
Instruction Set Architecture (ISA)

- Complete set of instructions used by a machine
- Abstract interface between the HW and lowest-level SW.
- An ISA includes the following …
  - Instructions and Instruction Formats
    - Data Types, Encodings, and Representations
    - Programmable Storage: Registers and Memory
    - Addressing Modes: to address Instructions and Data
    - Handling Exceptional Conditions (like division by zero)
- Examples (Versions) First Introduced in
  - Intel (8086, 80386, Pentium, ...) 1978
  - MIPS (MIPS I, II, III, IV, V) 1986
  - PowerPC (601, 604, …) 1993

The Instruction Set Architecture

- ISA is considered part of the SW
- Must be designed to survive changes in hardware technology, software technology, and application characteristic.
  - Is the agreed-upon interface between all the software that runs on the machine and the hardware that executes it.
- Advantages:
  - Different implementations of the same architecture
  - Easier to change than HW
  - Standardizes instructions, machine language bit patterns, etc.
- Disadvantage:
  - Sometimes prevents using new innovations

Instruction Set Architecture: Critical Interface



- Properties of a good abstraction
  - Lasts through many generations (portability)
  - Used in many different ways (generality)
  - Provides convenient functionality to higher levels
  - Permits an efficient implementation at lower levels

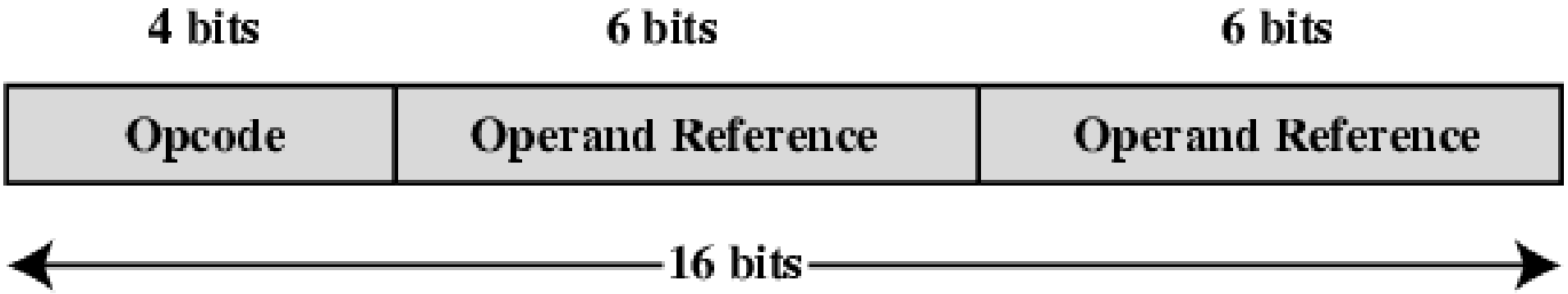
Elements of an Instruction

- Operation code (Op code)
  - Specify the operation (e.g., ADD, I/O)
- Source Operand reference
  - Operands that are input to the operation.
- Result Operand reference
  - Put the answer here
- Next Instruction Reference
  - Tells the processor where to fetch the next instruction

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



Where have all the Operands Gone?
Where is the next instruction to be fetched?

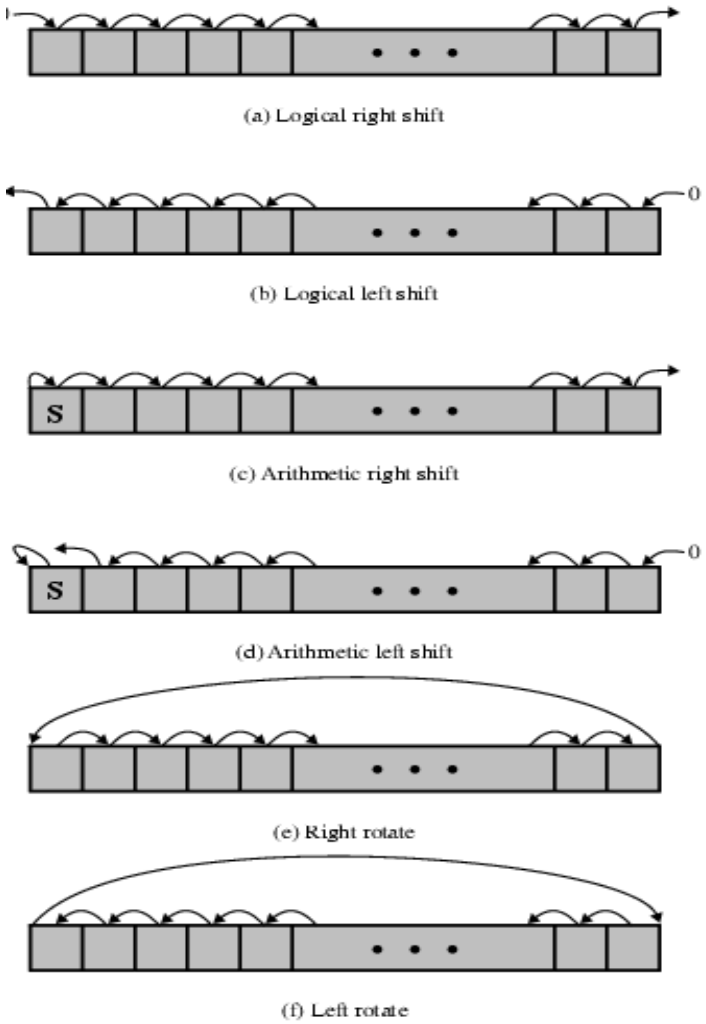
- Main memory (or virtual memory or cache)
- CPU register
- I/O device

Typical Instructions

Data Movement	Load (from memory) Store (to memory) memory-to-memory move register-to-register move input (from I/O device) output (to I/O device) push, pop (to/from stack)
Arithmetic	Data Types: (signed & unsigned) Integer (binary + decimal) (signed & unsigned) Floating Point Numbers Operations: Add, Subtract, Multiply, Divide
Logical	Not, and, or, set, clear
Shift	Arithmetic (& Logical) shift (left/right), rotate (left/right)
Control (Jump/Branch)	unconditional, conditional
Subroutine Linkage	call, return
Interrupt	trap, return
Synchronisation	test & set (atomic r-m-w)
String	search, compare, translate

Types of Operation

Shift and Rotate Operations



(a) Logical right shift

(b) Logical left shift

(c) Arithmetic right shift

(d) Arithmetic left shift

(e) Right rotate

(f) Left rotate

Types of Operation

- 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
  - For operating systems use

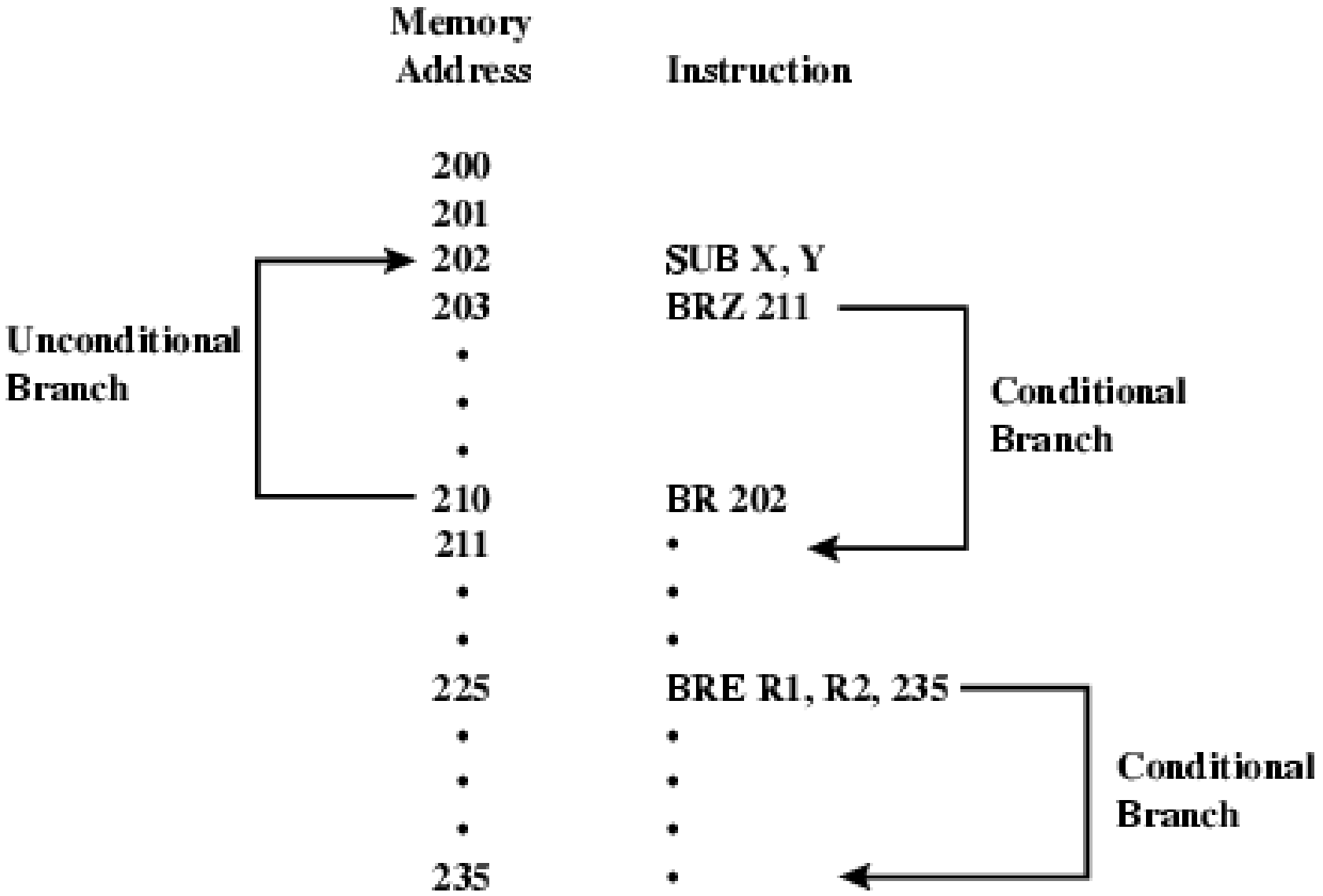
Transfer of Control

- Branch
  - e.g. ***BRZ X*** branch to x if result of (ADD,SUB,…) is zero
  - Uses condition bits register
  - See next slide
- Skip
  - e.g. increment and skip if zero ISZ

```
301
 .
 .
309      ISZ  R1
310      BR   301
311
```

  * eg. R1is set to -1000, the loop will be executed 1000 times
- Subroutine call
  - c.f. interrupt call

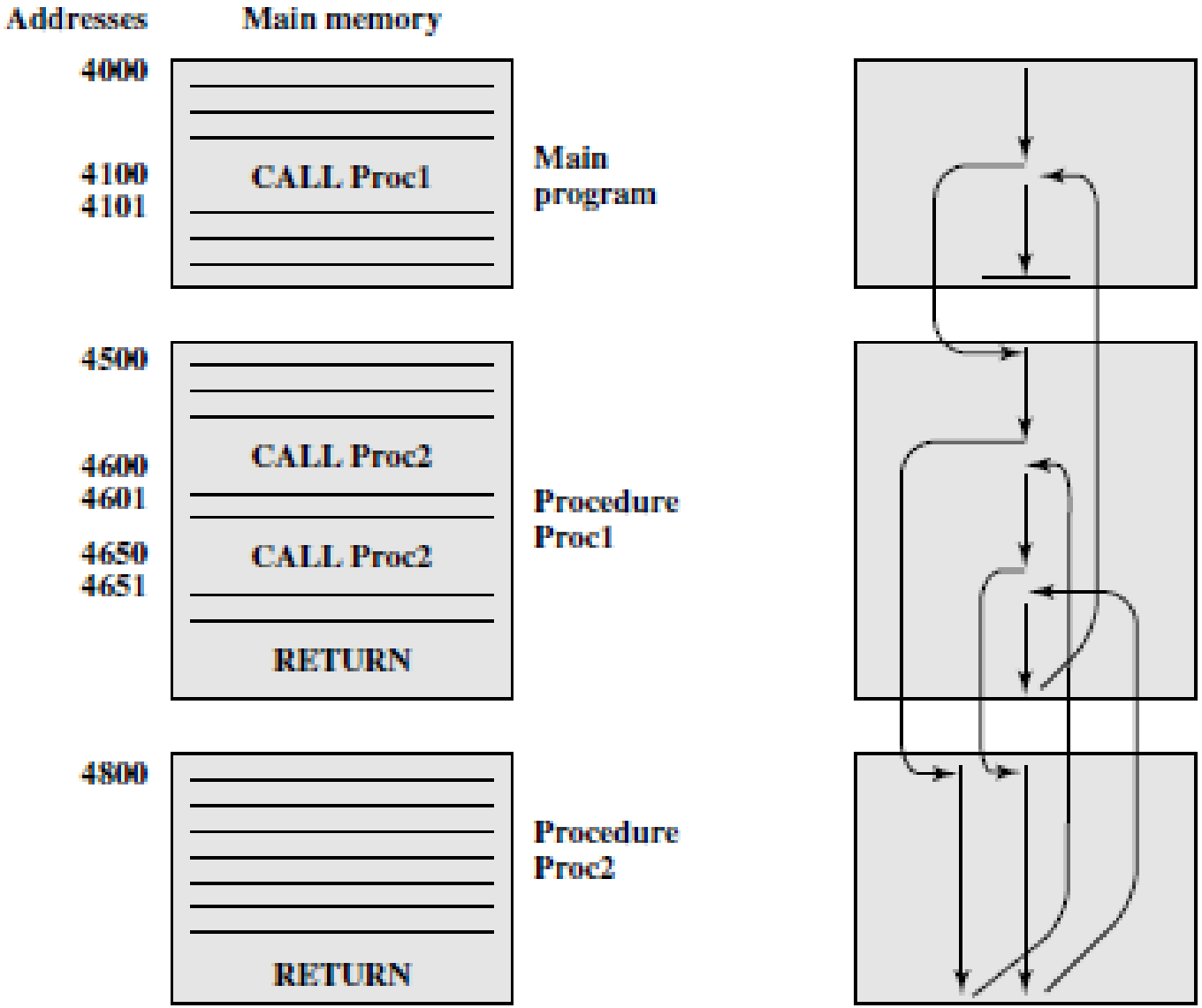
Branch Instruction



Procedure Calls Instructions

- Computer program that is incorporated with larger program.
- At any point in the program the procedure may be invoked, or ***called***
- When the procedure is executed, return to the point at which the call took place.
- Advantages:
  - Economy:
    - The same piece of Code can be used many times- **efficient use of storage space in the system**
  - Modularity
    - Allow large programming tasks to be divided into smaller units which **eases the programming task**

Procedure Calls Instructions

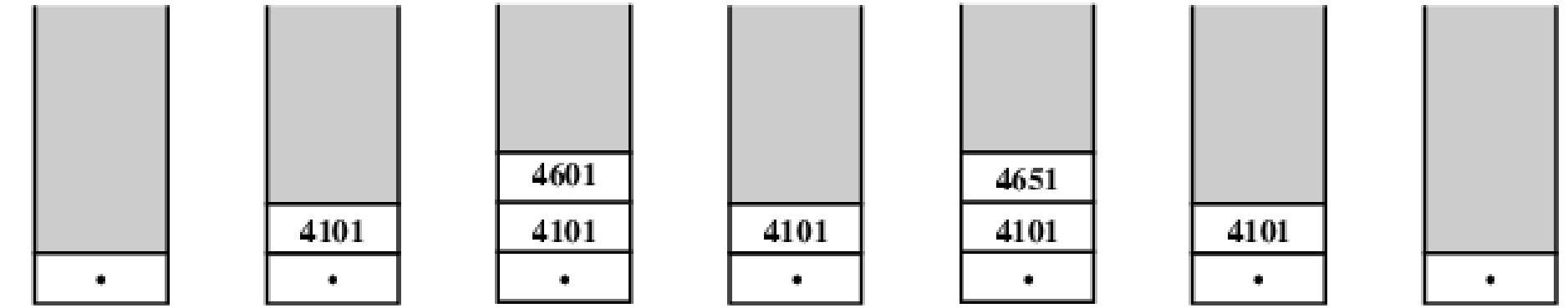


Procedure Calls Instructions

- Involves two basic instructions
  - Call: branch to the procedure location
  - Return: from the procedure to the place from which it was called

- ***Stack*** can be used to store the return address.

Use of Stack


4101
4601
4101
4101
4651
4101
4101
(a) Initial stack contents
(b) After CALL Proc1
(c) Initial CALL Proc2
(d) After RETURN
(e) After CALL Proc2
(f) After RETURN
(g) After RETURN

Types of Operand

- Addresses
- Numbers
  - Integer/floating point
- Characters
  - ASCII (IRA) etc.
- Logical Data
  - Bits or flags

Number of Addresses in Instruction

- 3 addresses
  - Operand 1, Operand 2, Result
  - ADD a,b,c (a = b + c;)

Instruction		Comment
SUB	Y, A, B	Y ← A − B
MPY	T, D, E	T ← D × E
ADD	T, T, C	T ← T + C
DIV	Y, Y, T	Y ← Y ÷ T

Number of Addresses

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

Instruction	Comment
MOVE Y, A	Y ← A
SUB Y, B	Y ← Y − B
MOVE T, D	T ← D
MPY T, E	T ← T × E
ADD T, C	T ← T + C
DIV Y, T	Y ← Y ÷ T

Number of Addresses

- 1 address
  - Implicit second address
  - Usually a register (accumulator)
  - ADD B (AC = AC + B)
  - Common on early machines

Instruction	Comment
LOAD D	AC ← D
MPY E	AC ← AC × E
ADD C	AC ← AC + C
STOR Y	Y ← AC
LOAD A	AC ← A
SUB B	AC ← AC − B
DIV Y	AC ← AC ÷ Y
STOR Y	Y ← AC

Number of Addresses

- 0 (zero) addresses
  - —Applicable to a special memory organization called ***Stack***
  - —Stack is known location
  - —Often at least the top two stack elements are in processor registers
    - ADD
    - All addresses implicit

Number of Addresses

- 4 addresses
  - —Operand 1, Operand 2, Result, and next instruction
  - —Not common
  - —Needs very long words to hold everything

Number of Addresses

Number of Addresses	Symbolic Representation	Interpretation
3	OP A, B, C	A ← B OP C
2	OP A, B	A ← A OP B
1	OP A	AC ← AC OP A
0	OP	T ← (T − 1) OP T

How Many Addresses

- More addresses
  - More complex (powerful?) instructions
  - More registers
    - Inter-register operations are quicker
  - Fewer instructions per program
- Fewer addresses
  - Less complex (powerful?) instructions
  - More instructions per program
  - Faster fetch/execution of instructions
- Most processor designs involve a variety of instruction formats.

Fundamental Issues in Instruction Set Design

- Operation repertoire
  - How many ops?
  - What can they do?
  - How complex are they?
- Data types
  - The data type that the processor can deal with
  - E.g., Pentium can deal wit data types of:
    - Byte, 8 bits
    - Word, 16 bits
    - Doubleword, 32 bits
    - Quadword, 64 bits
    - Other data type…
- Instruction formats
  - Length of op code field
  - Number of addresses

Fundamental Issues in Instruction Set Design

- Registers
  - Number of CPU registers available
  - Which operations can be performed on which registers?
- Addressing modes (later…)
- RISC v CISC

Types of Operand

- Addresses
- Numbers
  - Integer/floating point
- Characters
  - ASCII etc.
- Logical Data
  - Bits or flags

Where have all the Operands Gone?
Where is the next instruction to be fetched?

- Main memory (or virtual memory or cache)
- CPU register
- I/O device

Code Example

- In a CPU with registers s0 – s4, interpret the following code:

```
mov s0, 5
mov s1, -3
mul  s2, s1, s0
mul  s0, s0
mul  s1, s1
sub  s1, s0
add  s2, s1
```

Summary points

- What is an instruction set architecture (ISA)?
- What are the instruction types?
- What are the basic components of an instruction?
- What is the stack?