William Stallings
Computer Organization
and Architecture
8th Edition

Chapter 10

Instruction Sets:

Characteristics and Functions

What is an Instruction Set?

- The complete collection of instructions that are understood by a CPU
- Machine Code
- Binary
- Usually represented by assembly codes

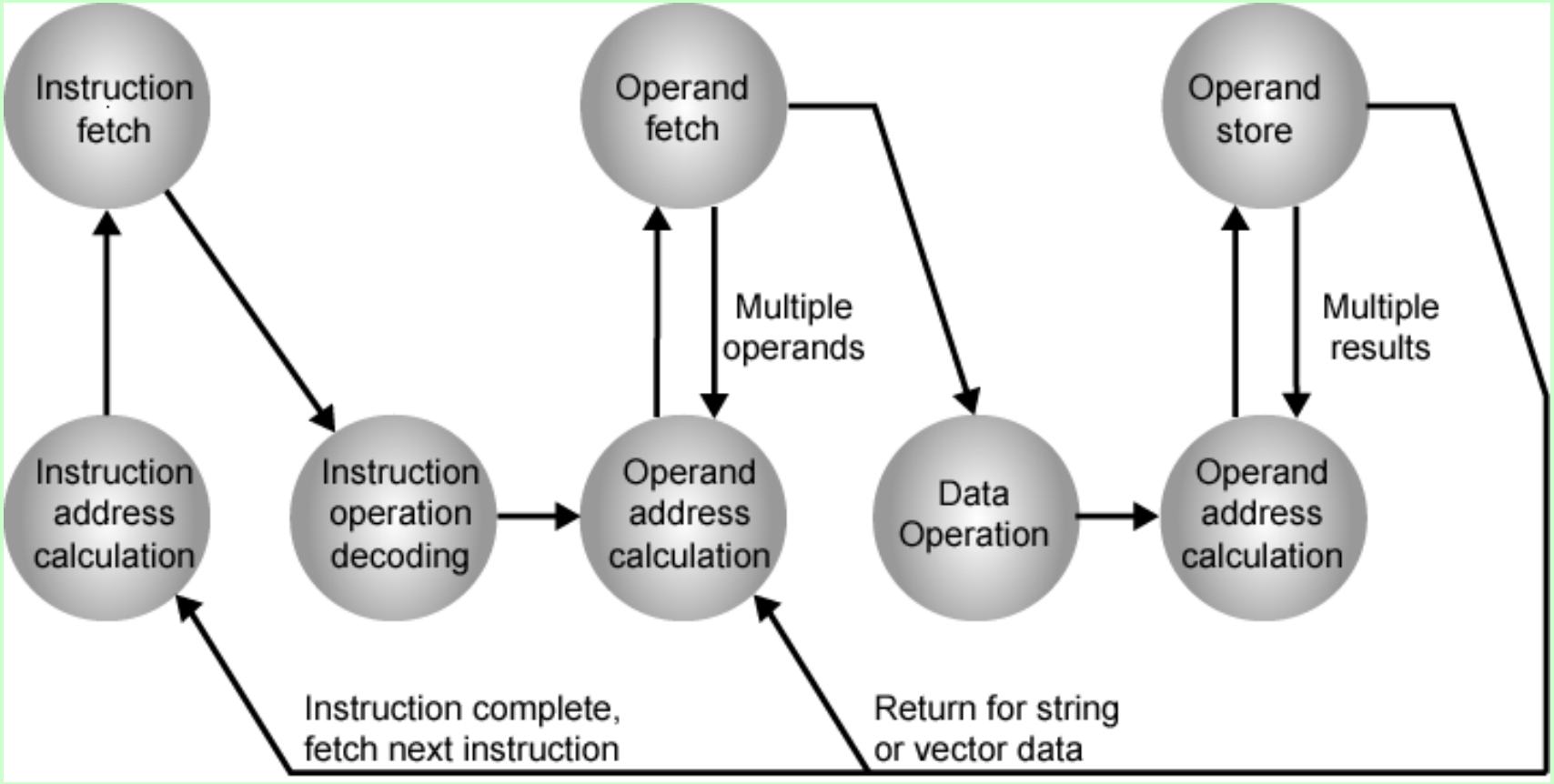
Elements of an Instruction

- 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...

Where have all the Operands Gone?

- Long time passing….
- (If you don’t understand, you’re too young!)
- Main memory (or virtual memory or cache)
- CPU register
- I/O device

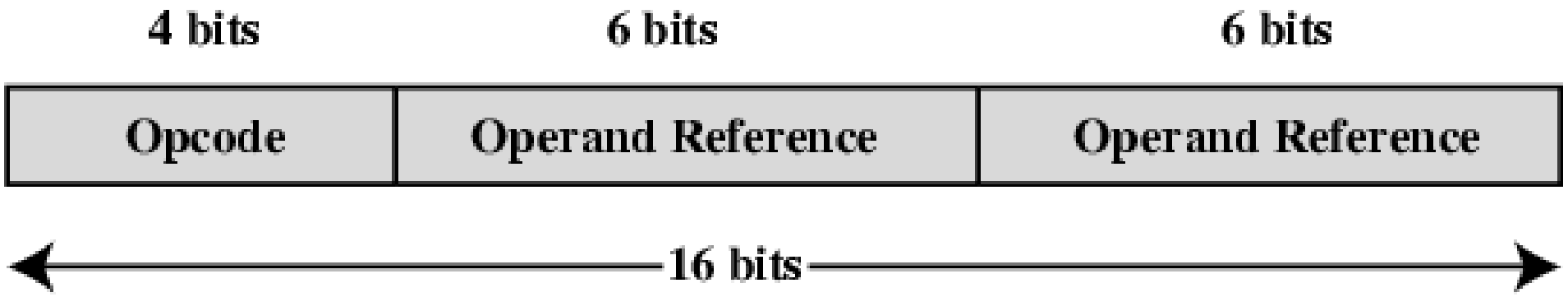
Instruction Cycle State Diagram



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
- Data storage (main memory)
- Data movement (I/O)
- Program flow control

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

Number of Addresses (d)

- 0 (zero) addresses
  - All addresses implicit
  - Uses a stack
  - e.g. push a
  - push b
  - add
  - pop c

  - c = a + b

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

Design Decisions (1)

- Operation repertoire
  - How many ops?
  - What can they do?
  - How complex are they?
- Data types
- Instruction formats
  - Length of op code field
  - Number of addresses

Design Decisions (2)

- 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
- (Aside: Is there any difference between numbers and characters? Ask a C programmer!)

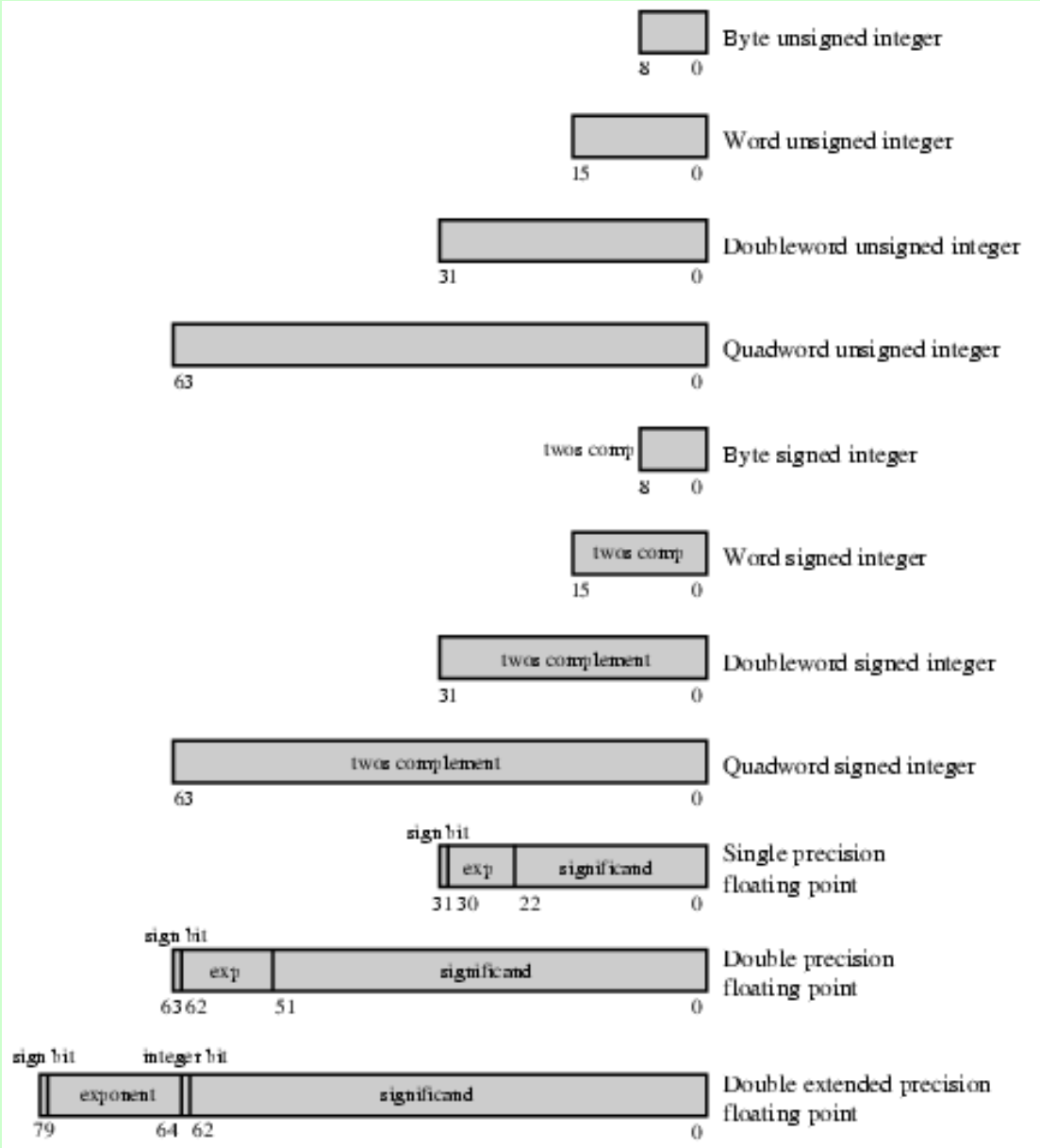
x86 Data Types

- 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

SMID Data Types

- Integer types
  - Interpreted as bit field or integer
- Packed byte and packed byte integer
  - Bytes packed into 64-bit quadword or 128-bit double quadword
- Packed word and packed word integer
  - 16-bit words packed into 64-bit quadword or 128-bit double quadword
- Packed doubleword and packed doubleword integer
  - 32-bit doublewords packed into 64-bit quadword or 128-bit double quadword
- Packed quadword and packed qaudword integer
  - Two 64-bit quadwords packed into 128-bit double quadword
- Packed single-precision floating-point and packed double-precision floating-point
  - Four 32-bit floating-point or two 64-bit floating-point values packed into a 128-bit double quadword

x86 Numeric Data Formats

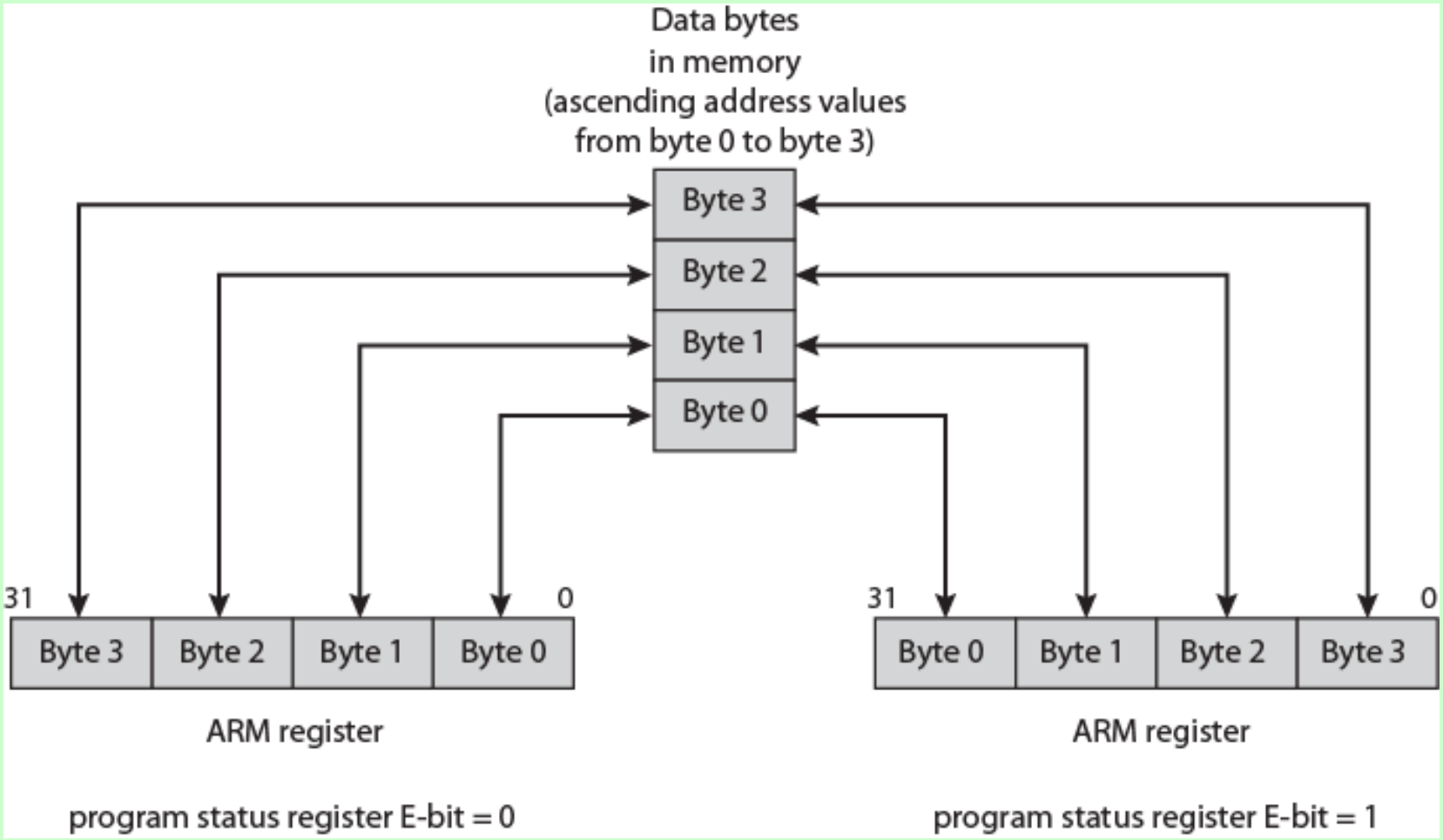


ARM Data Types

- 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 bytesAlignment 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
- 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

ARM Endian Support

- E-bit in system control register
- Under program control



Types of Operation

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

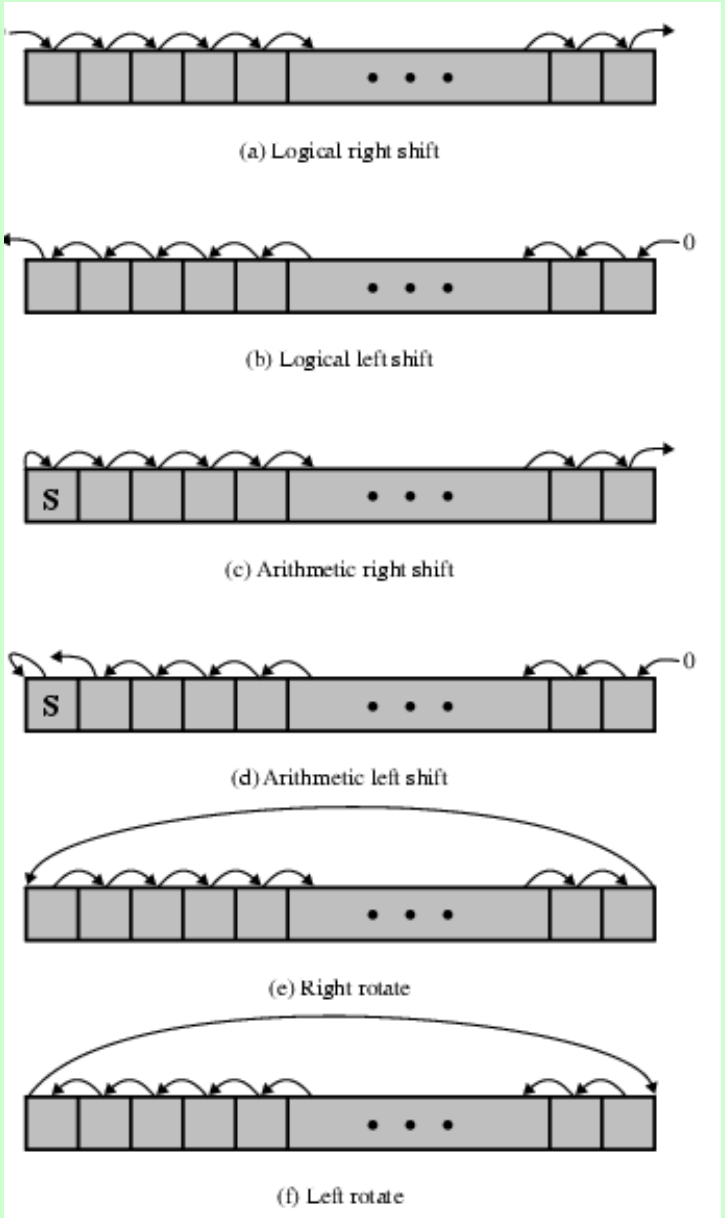
Data Transfer

- 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)

Shift and Rotate Operations



Logical

- Bitwise operations
- AND, OR, NOT

Conversion

- 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)

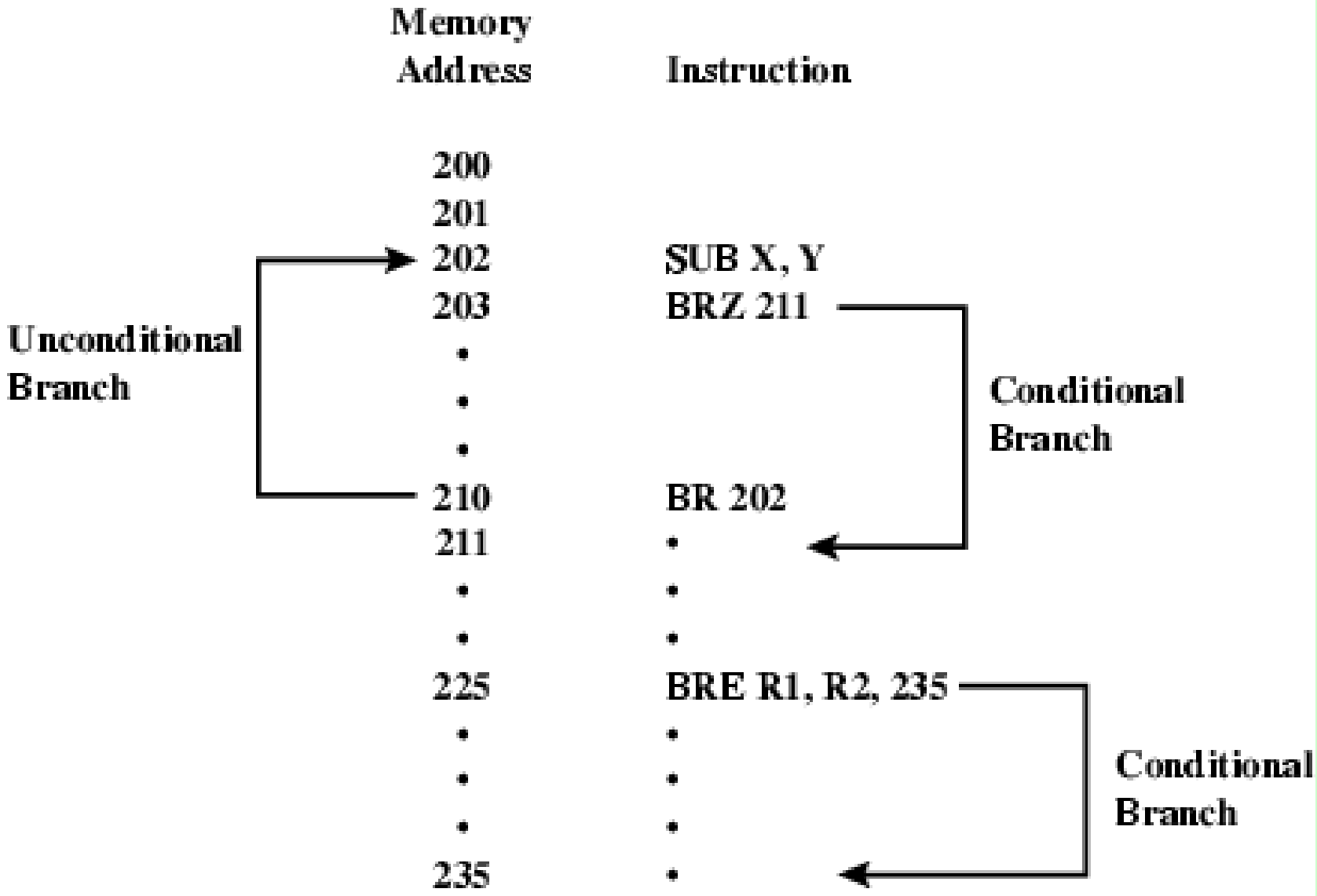
Systems Control

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

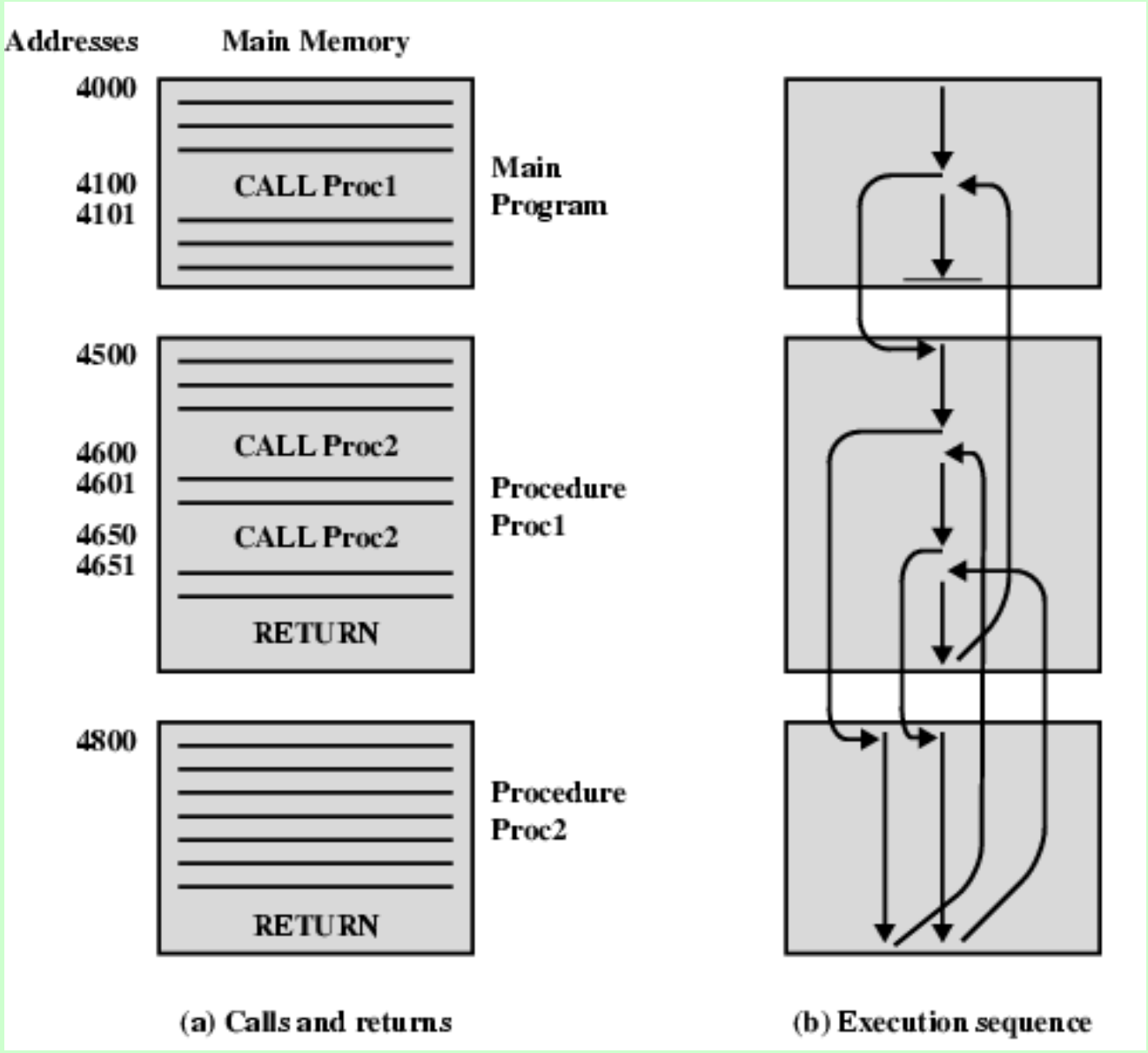
Transfer of Control

- 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

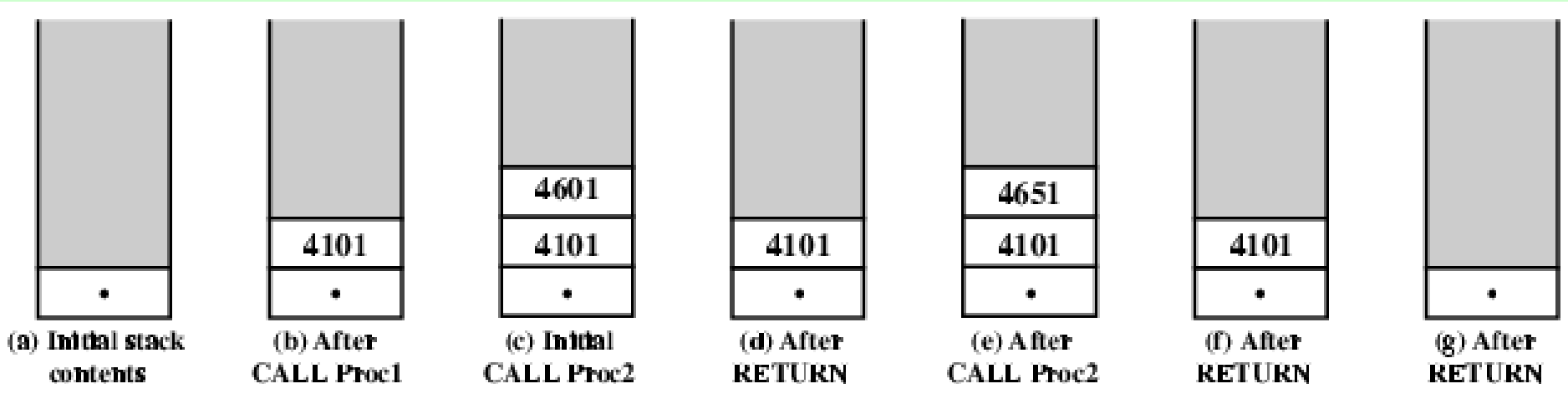
Branch Instruction



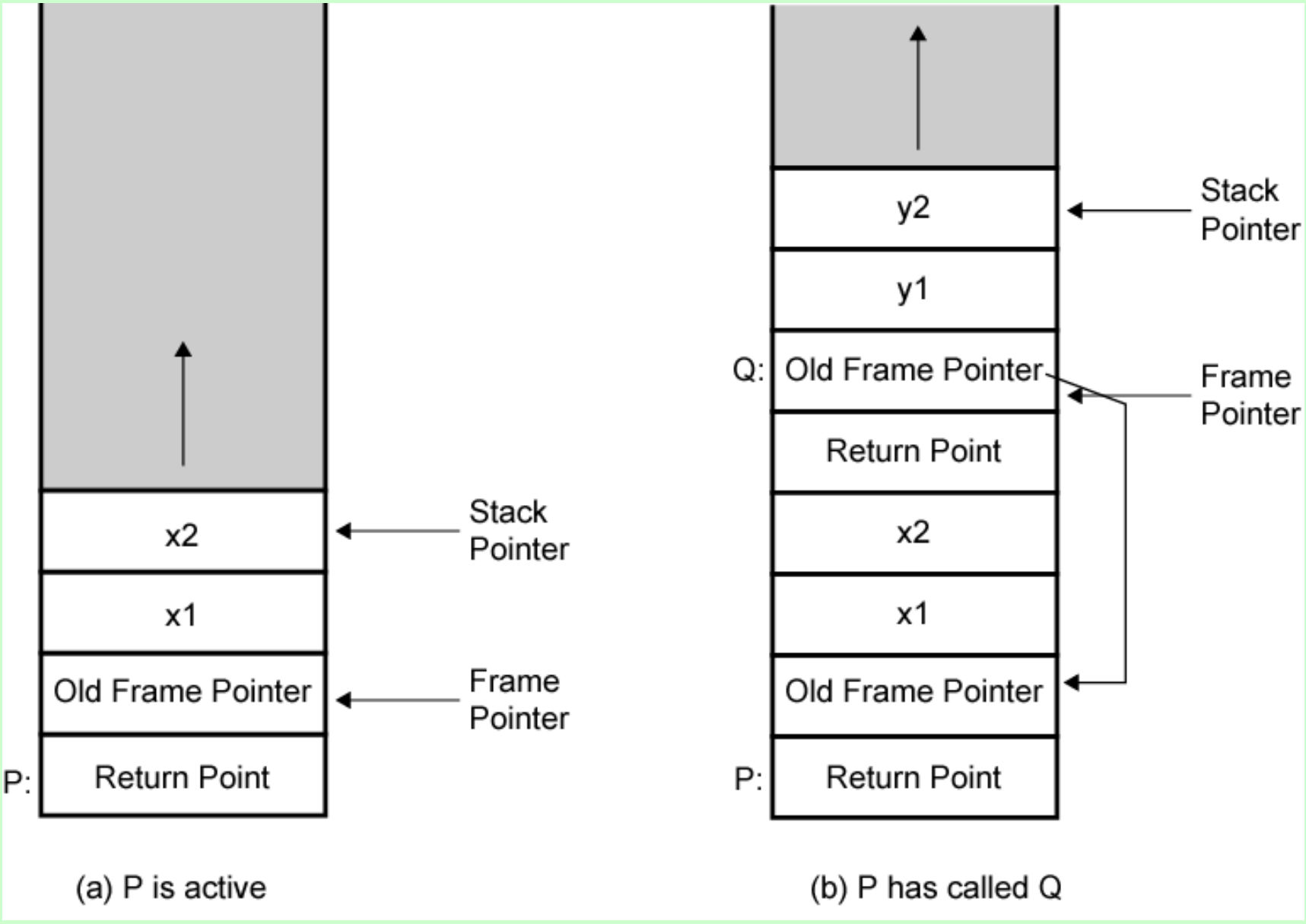
Nested Procedure Calls



Use of Stack


4601
4101
4101
4651
4101
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

Stack Frame Growth Using Sample Procedures P and Q



Exercise For Reader

- Find out about instruction set for Pentium and ARM
- Start with Stallings
- Visit web sites

Byte Order
(A portion of chips?)

- What order do we read numbers that occupy more than one byte
- e.g. (numbers in hex to make it easy to read)
- 12345678 can be stored in 4x8bit locations as follows

Byte Order (example)

Address	Value (1)	Value(2)
184	12	78
185	34	56
186	56	34
186	78	12

- i.e. read top down or bottom up?

Byte Order Names

- The problem is called Endian
- The system on the left has the least significant byte in the lowest address
- This is called big-endian
- The system on the right has the least significant byte in the highest address
- This is called little-endian

Example of C Data Structure

```
struct{
   int     a;     //0x1112_1314                    word
   int     pad;   //
   double  b;     //0x2122_2324_2526_2728          doubleword
   char*   c;     //0x3132_3334                    word
   char    d[7];  //'A','B','C','D','E','F','G'   byte array
   short   e;     //0x5152                         halfword
   int     f;     //0x6161_6364                    word
} s;
```

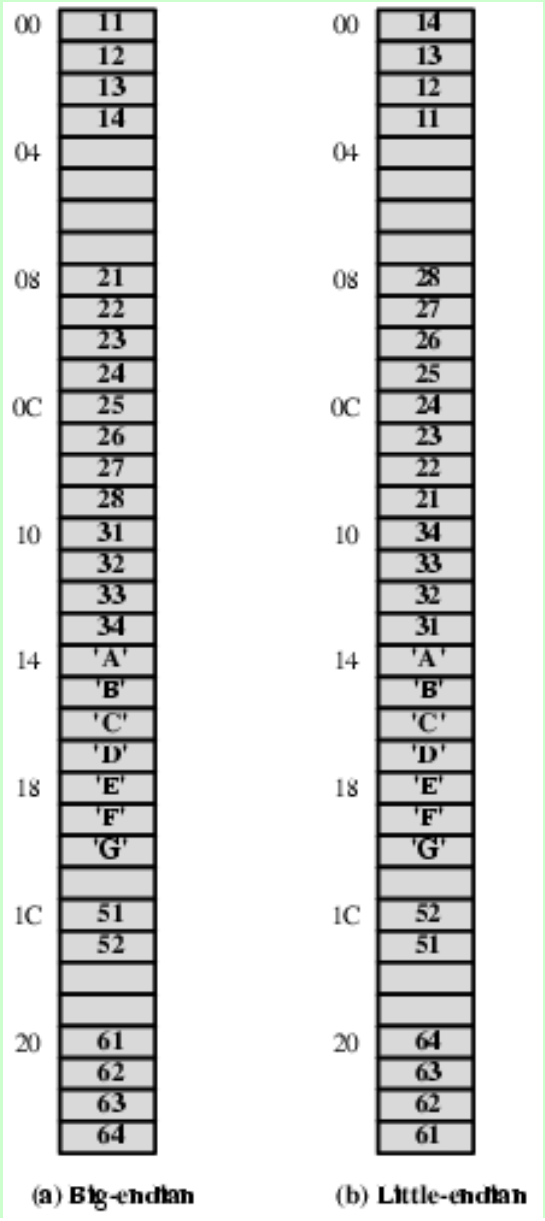
Big-endian address mapping

Byte Address								
00	**11** 00	**12** 01	**13** 02	**14** 03	04	05	06	07
08	**21** 08	**22** 09	**23** 0A	**24** 0B	**25** 0C	**26** 0D	**27** 0E	**28** 0F
10	**31** 10	**32** 11	**33** 12	**34** 13	**'A'** 14	**'B'** 15	**'C'** 16	**'D'** 17
18	**'E'** 18	**'F'** 19	**'G'** 1A	1B	**51** 1C	**52** 1D	1E	1F
20	**61** 20	**62** 21	**63** 22	**64** 23				

Little-endian address mapping

								Byte Address
				11 03	**12** 02	**13** 01	**14** 00	00
21 0F	**22** 0E	**23** 0D	**24** 0C	**25** 0B	**26** 0A	**27** 09	**28** 08	08
'D' 17	**'C'** 16	**'B'** 15	**'A'** 14	**31** 13	**32** 12	**33** 11	**34** 10	10
		51 1D	**52** 1C		**'G'** 1A	**'F'** 19	**'E'** 18	18
				61 23	**62** 22	**63** 21	**64** 20	20

Alternative View of Memory Map



Standard...What Standard?

- Pentium (x86), VAX are little-endian
- IBM 370, Moterola 680x0 (Mac), and most RISC are big-endian
- Internet is big-endian
  - Makes writing Internet programs on PC more awkward!
  - WinSock provides htoi and itoh (Host to Internet & Internet to Host) functions to convert