Chapter 10
Instruction Sets:
Characteristics and Functions

2020.6
Howon Kim

- 정보보호 및 지능형 IoT연구실 - http://infosec.pusan.ac.kr
- 부산대 지능형융합보안대학원 - http://aisec.pusan.ac.kr

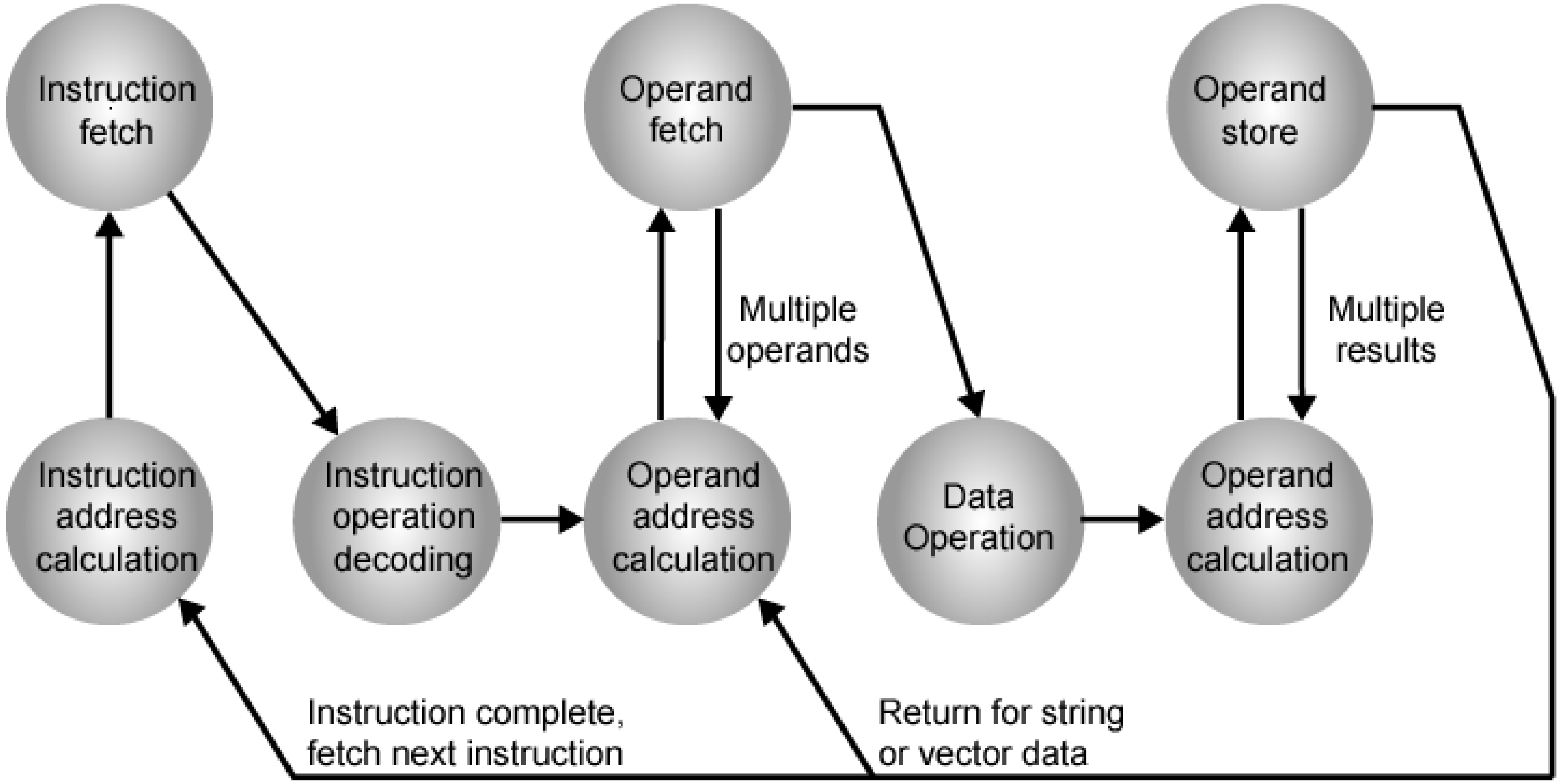
What is an Instruction Set?

- 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...
- Ref (wikipedia)
  - In mathematics, an **operand** is one of the inputs (arguments) of an operator
  - For instance, in 3 + 6 = 9, '+' is the operator and '3' and '6' are the operands
  - In computer programming languages, the definitions of operator and **operand** are almost the same as in mathematics.

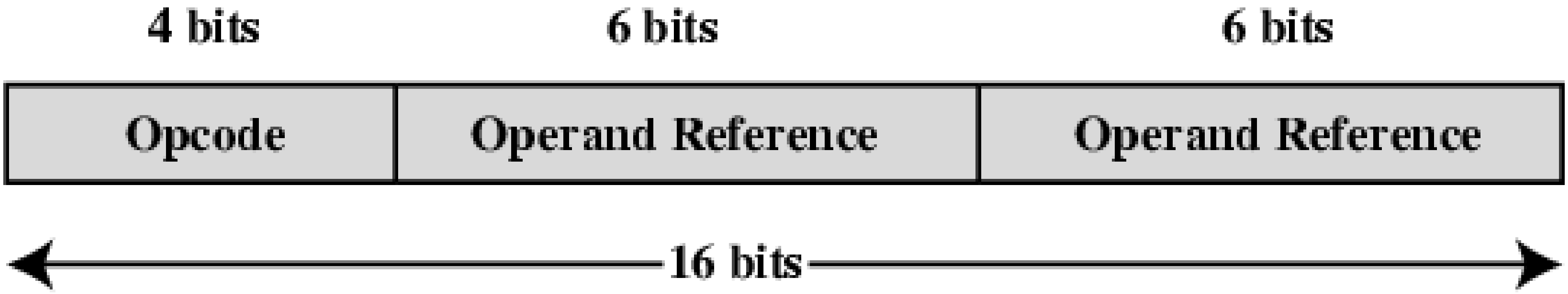
Instruction Cycle State Diagram


Instruction fetch
Operand fetch
Operand store
Multiple operands
Multiple results
Instruction address calculation
Instruction operation decoding
Operand address calculation
Data Operation
Operand address calculation
Instruction complete, fetch next instruction
Return for string or vector data

Instruction Representation

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

Simple Instruction Format



4 bits

6 bits

6 bits

Opcode

Operand Reference

Operand Reference

16 bits

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 long words to hold everything

Number of Addresses (b)

- 2 addresses
  - One address is used twice as operand and result
    - a = a + b
  - Reduces length of instruction
  - May require 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 are implicitly interpreted
  - 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)

- In operation code (OP code)
  - How many ops?
  - What can they do?
  - How complex they are ?
- 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
- RISC vs. CISC

Types of Operand

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

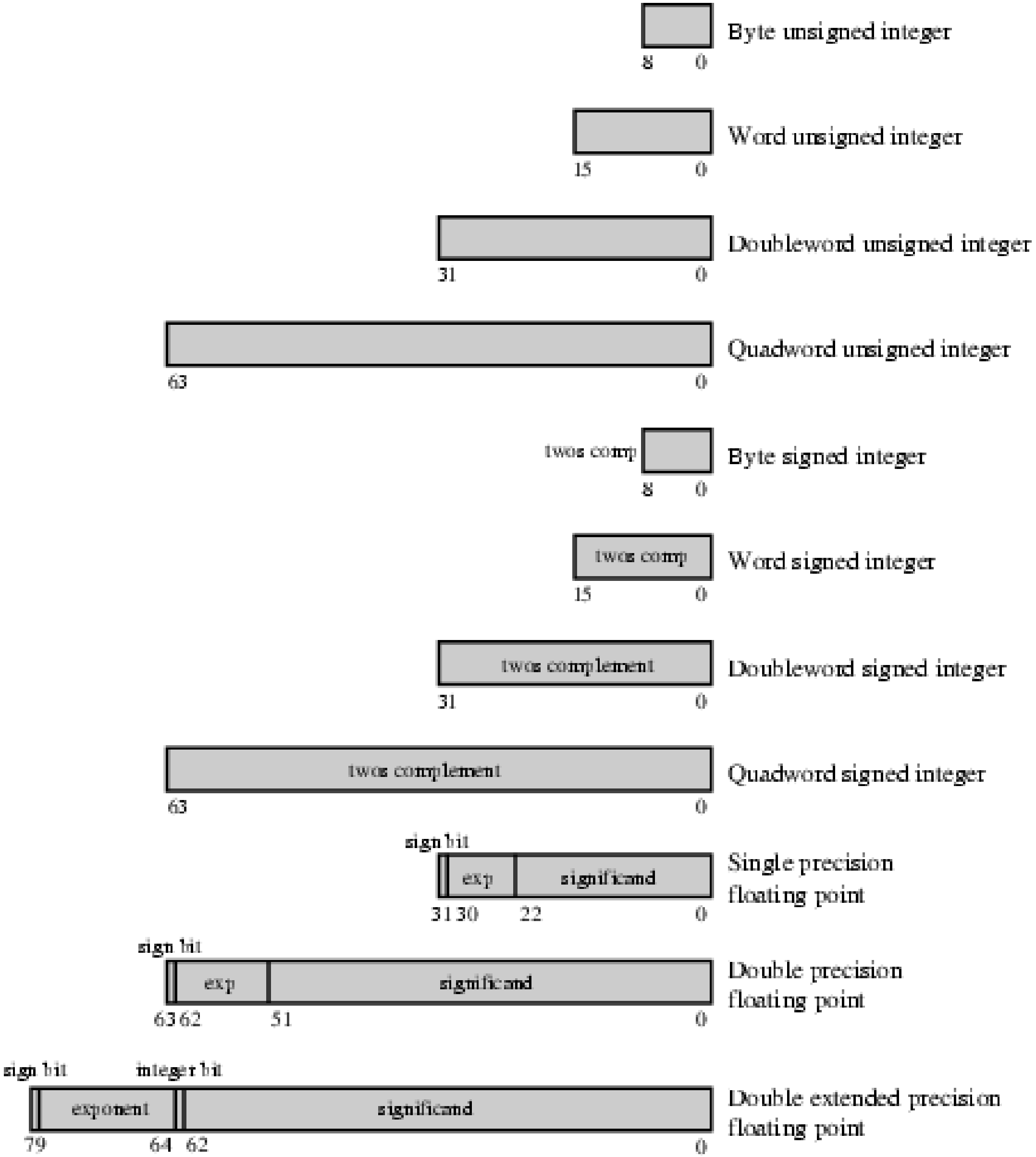
Pentium Data Types

- Byte : 8 bit
- Word : 16 bit
- Double word : 32 bit
- Quad word : 64 bit
- Addressing is by 8 bit unit
    - A 32 bit double word is read at addresses divisible by 4

Specific Data Types

- General - arbitrary binary contents
- Integer - single binary value
- Ordinal - unsigned integer
- Unpacked BCD - One digit per byte
- Packed BCD - 2 BCD digits per byte
- Near Pointer - 32 bit offset within segment
- Bit field
- Byte String
- Floating Point

Pentium Numeric Data Formats



PowerPC Data Types

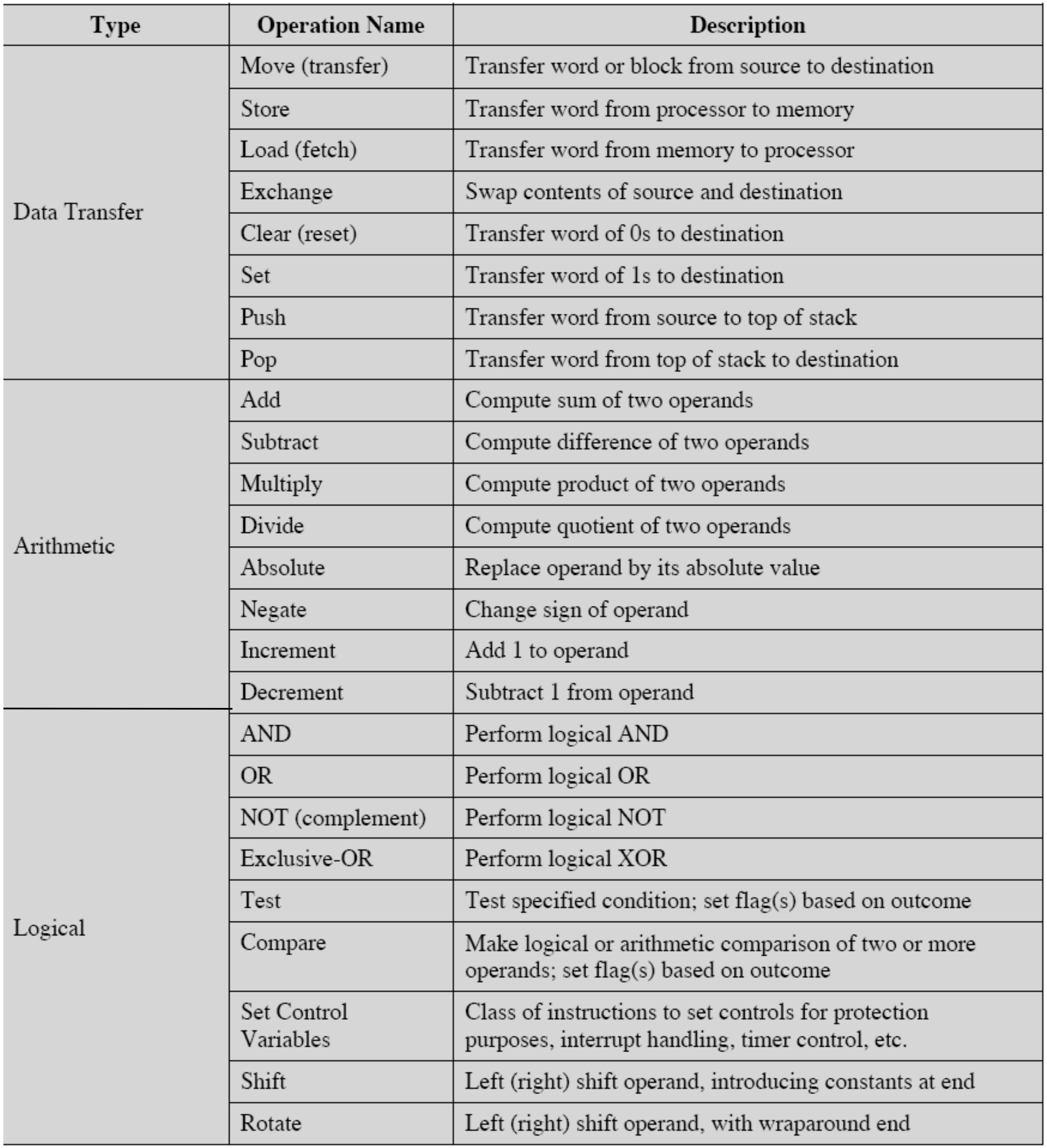
- 8 (byte), 16 (halfword), 32 (word) and 64 (doubleword) length data types
- Some instructions need operand aligned on 32 bit boundary
- Can be big- or little-endian
- Fixed point processor recognises:
  - Unsigned byte, unsigned halfword, signed halfword, unsigned word, signed word, unsigned doubleword, byte string (<128 bytes)
- Floating point
  - IEEE 754
  - Single or double precision

Types of Operation

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

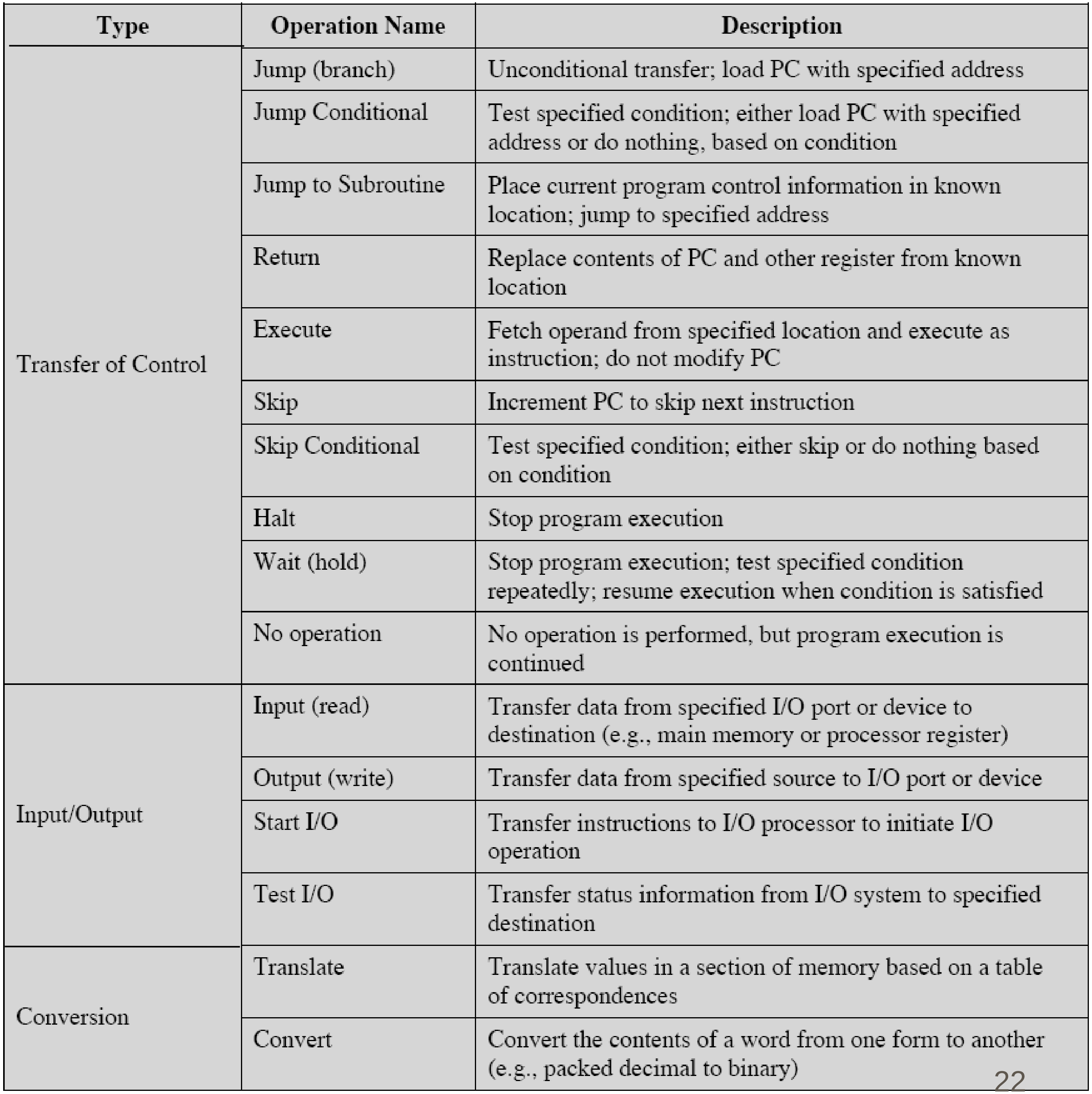
Types of Operation

Table 10.3 Common Instruction Set Operations (page 1 of 2)



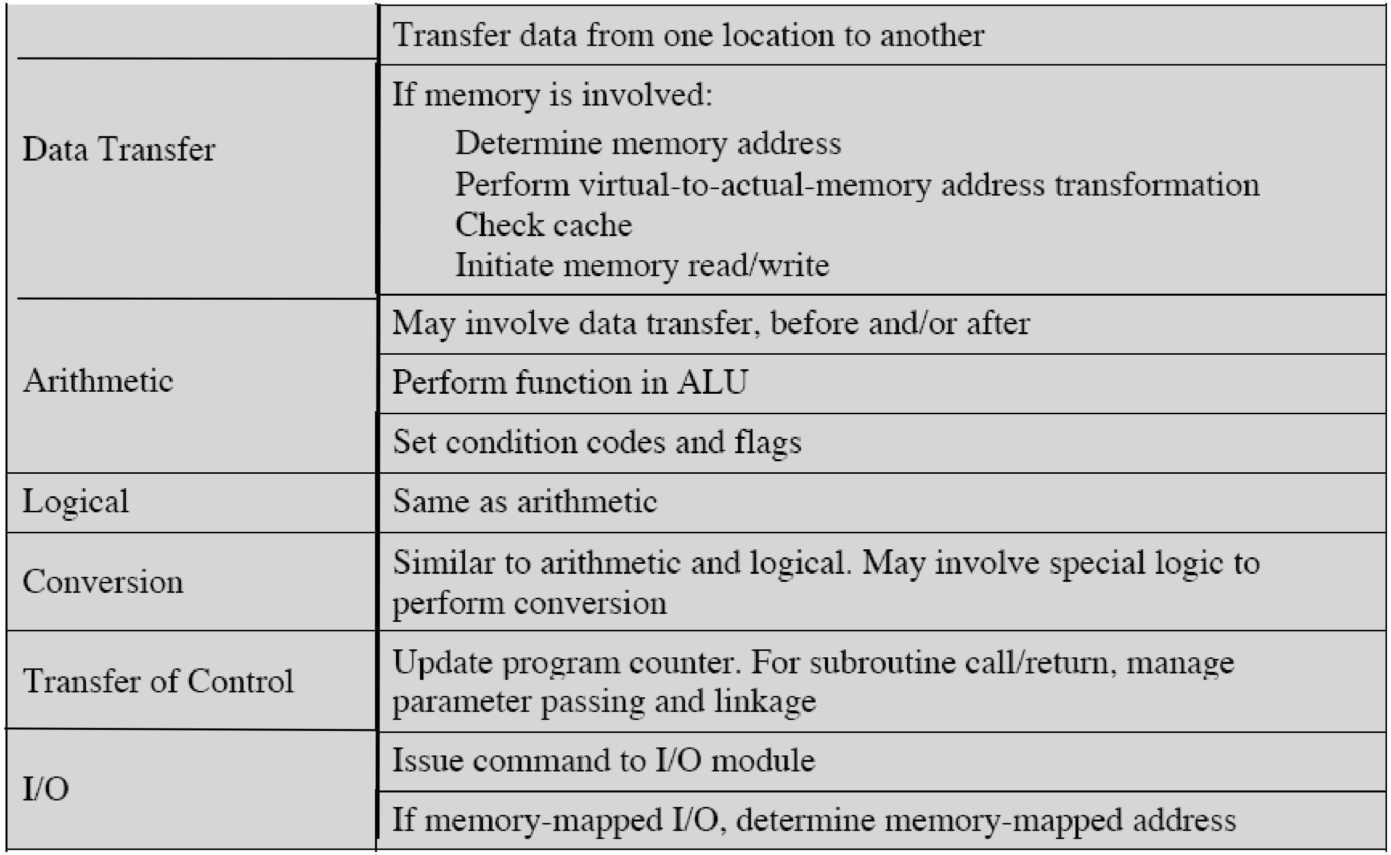
Type	Operation Name	Description
Data Transfer	Move (transfer)	Transfer word or block from source to destination
	Store	Transfer word from processor to memory
	Load (fetch)	Transfer word from memory to processor
	Exchange	Swap contents of source and destination
	Clear (reset)	Transfer word of 0s to destination
	Set	Transfer word of 1s to destination
	Push	Transfer word from source to top of stack
	Pop	Transfer word from top of stack to destination
Arithmetic	Add	Compute sum of two operands
	Subtract	Compute difference of two operands
	Multiply	Compute product of two operands
	Divide	Compute quotient of two operands
	Absolute	Replace operand by its absolute value
	Negate	Change sign of operand
	Increment	Add 1 to operand
	Decrement	Subtract 1 from operand
Logical	AND	Perform logical AND
	OR	Perform logical OR
	NOT (complement)	Perform logical NOT
	Exclusive-OR	Perform logical XOR
	Test	Test specified condition; set flag(s) based on outcome
	Compare	Make logical or arithmetic comparison of two or more operands; set flag(s) based on outcome
	Set Control Variables	Class of instructions to set controls for protection purposes, interrupt handling, timer control, etc.
	Shift	Left (right) shift operand, introducing constants at end
	Rotate	Left (right) shift operand, with wraparound end

Types of Operation



Type	Operation Name	Description
Transfer of Control	Jump (branch)	Unconditional transfer; load PC with specified address
	Jump Conditional	Test specified condition; either load PC with specified address or do nothing, based on condition
	Jump to Subroutine	Place current program control information in known location; jump to specified address
	Return	Replace contents of PC and other register from known location
	Execute	Fetch operand from specified location and execute as instruction; do not modify PC
	Skip	Increment PC to skip next instruction
	Skip Conditional	Test specified condition; either skip or do nothing based on condition
	Halt	Stop program execution
	Wait (hold)	Stop program execution; test specified condition repeatedly; resume execution when condition is satisfied
	No operation	No operation is performed, but program execution is continued
Input/Output	Input (read)	Transfer data from specified I/O port or device to destination (e.g., main memory or processor register)
	Output (write)	Transfer data from specified source to I/O port or device
	Start I/O	Transfer instructions to I/O processor to initiate I/O operation
	Test I/O	Transfer status information from I/O system to specified destination
Conversion	Translate	Translate values in a section of memory based on a table of correspondences
	Convert	Convert the contents of a word from one form to another (e.g., packed decimal to binary)

Types of Operation



Data Transfer	Transfer data from one location to another
	If memory is involved: Determine memory address Perform virtual-to-actual-memory address transformation Check cache Initiate memory read/write
Arithmetic	May involve data transfer, before and/or after
	Perform function in ALU
	Set condition codes and flags
Logical	Same as arithmetic
Conversion	Similar to arithmetic and logical. May involve special logic to perform conversion
Transfer of Control	Update program counter. For subroutine call/return, manage parameter passing and linkage
I/O	Issue command to I/O module
	If memory-mapped I/O, determine memory-mapped address

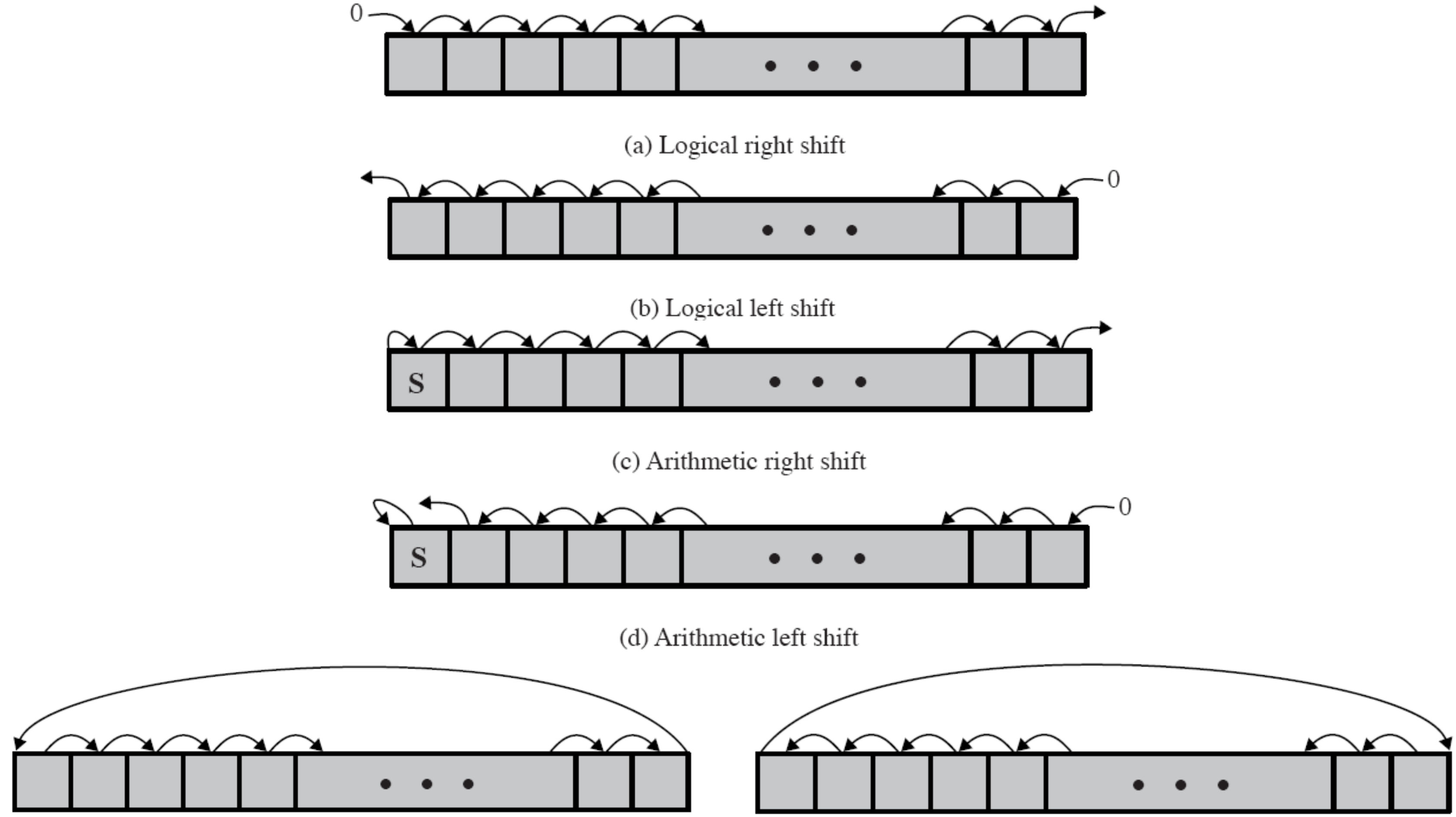
Data Transfer

- 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



(a) Logical right shift

(b) Logical left shift

(c) Arithmetic right shift

(d) Arithmetic left shift

(e) Right rotate

(f) Left rotate

Logical

- Bitwise operations
- AND, OR, NOT

Conversion

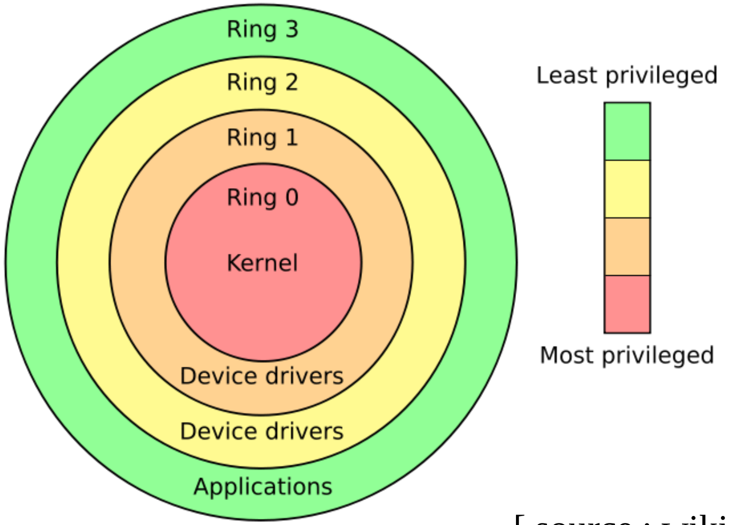
- E.g. Binary to Decimal

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+
  - Most privileged mode, Kernel mode
- For operating systems use



[source : wikipedia]

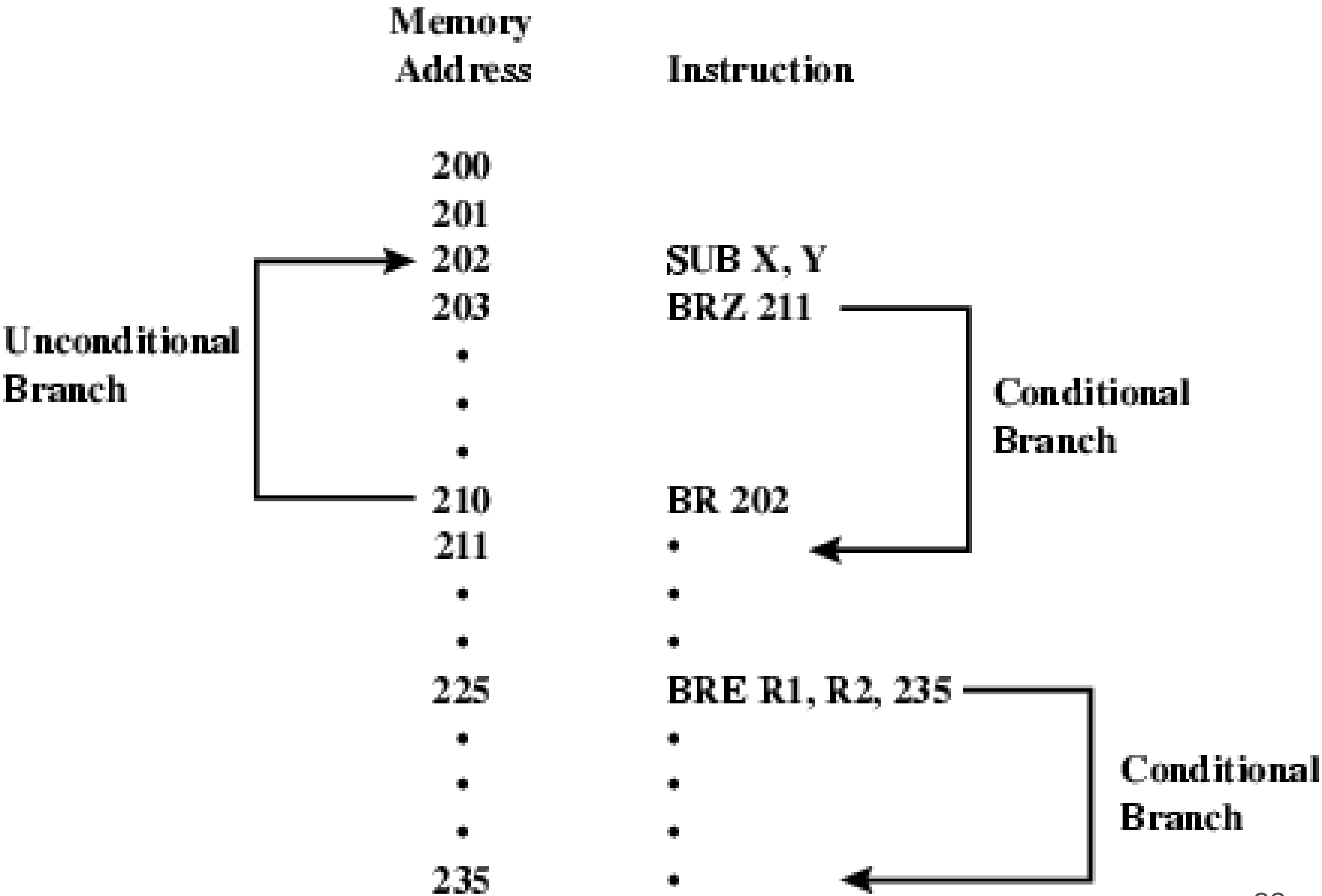
Transfer of Control

- Branch
  - e.g. branch to x if result is zero
- Skip
  - e.g. increment and skip if zero
  - ISZ Register1
    - **ISZ**(**I**ncrement operand and **s**kip next instruction if the result is **z**ero)
  - Branch xxxx
  - ADD A
- Subroutine call
  - c.f. interrupt call

< Example of loop >

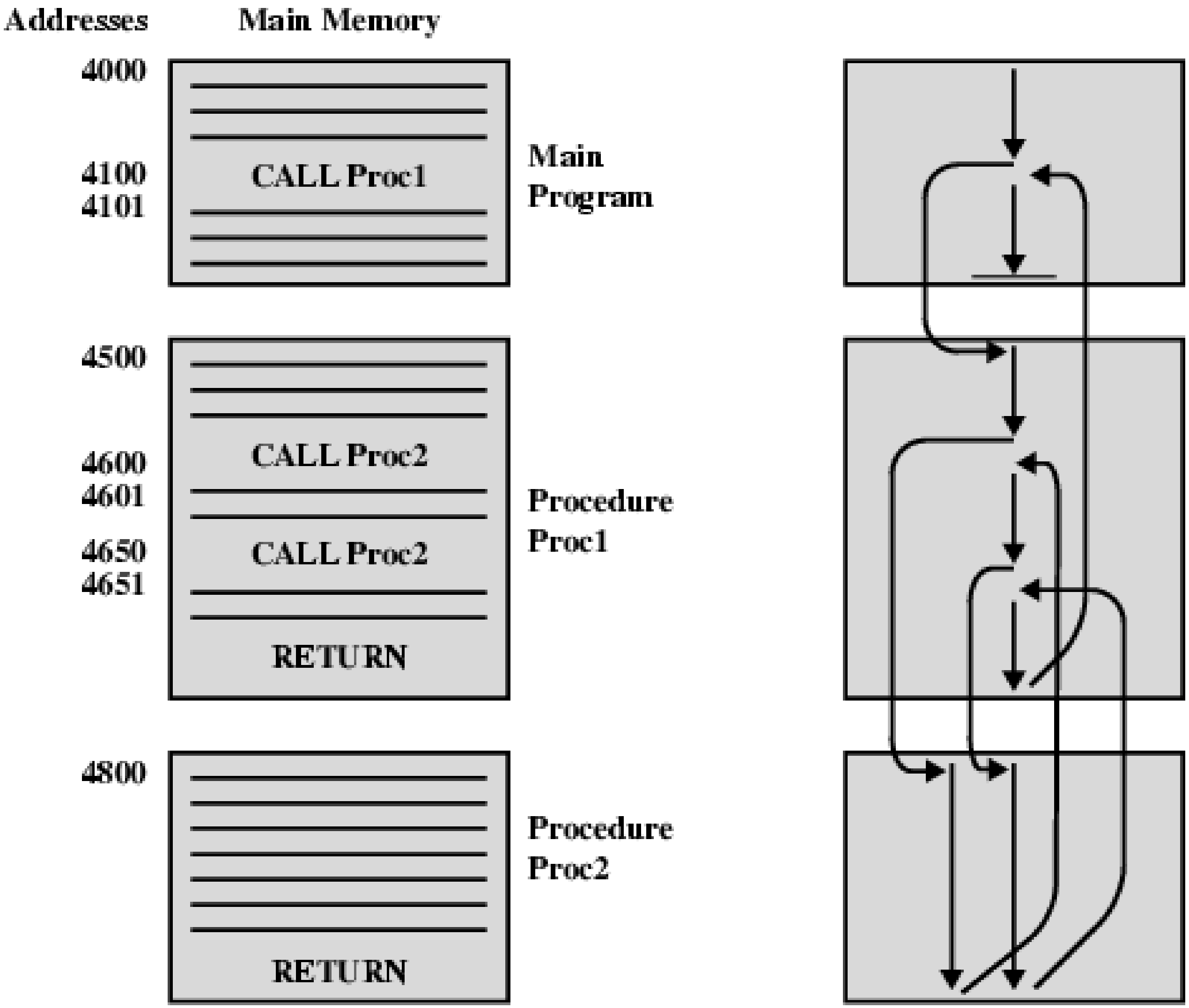
```
301
…
309 ISZ R1 // initially set to negative value
310 BR 301
311 // if R1 is zero, then skip next instruction [BR 301]
        //  and code [311] will be executed,
        // otherwise, loop is executed (i.e., go 301)
```

Branch Instruction



Memory Address	Instruction
200	
201	
202	SUB X, Y
203	BRZ 211
•	
•	
•	
210	BR 202
211	•
•	•
•	•
225	BRE R1, R2, 235
•	•
•	•
•	•
235	•

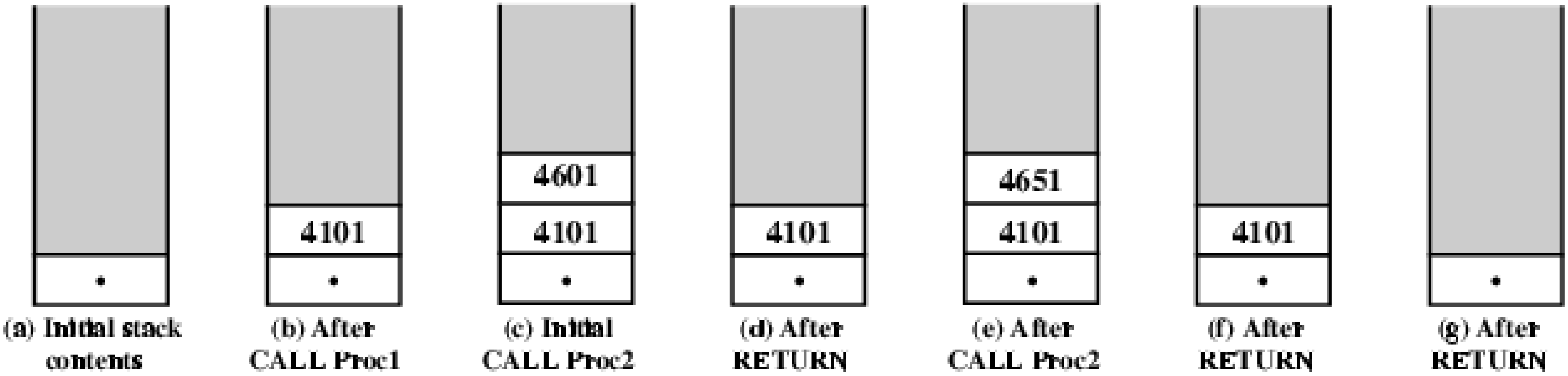
Nested Procedure Calls



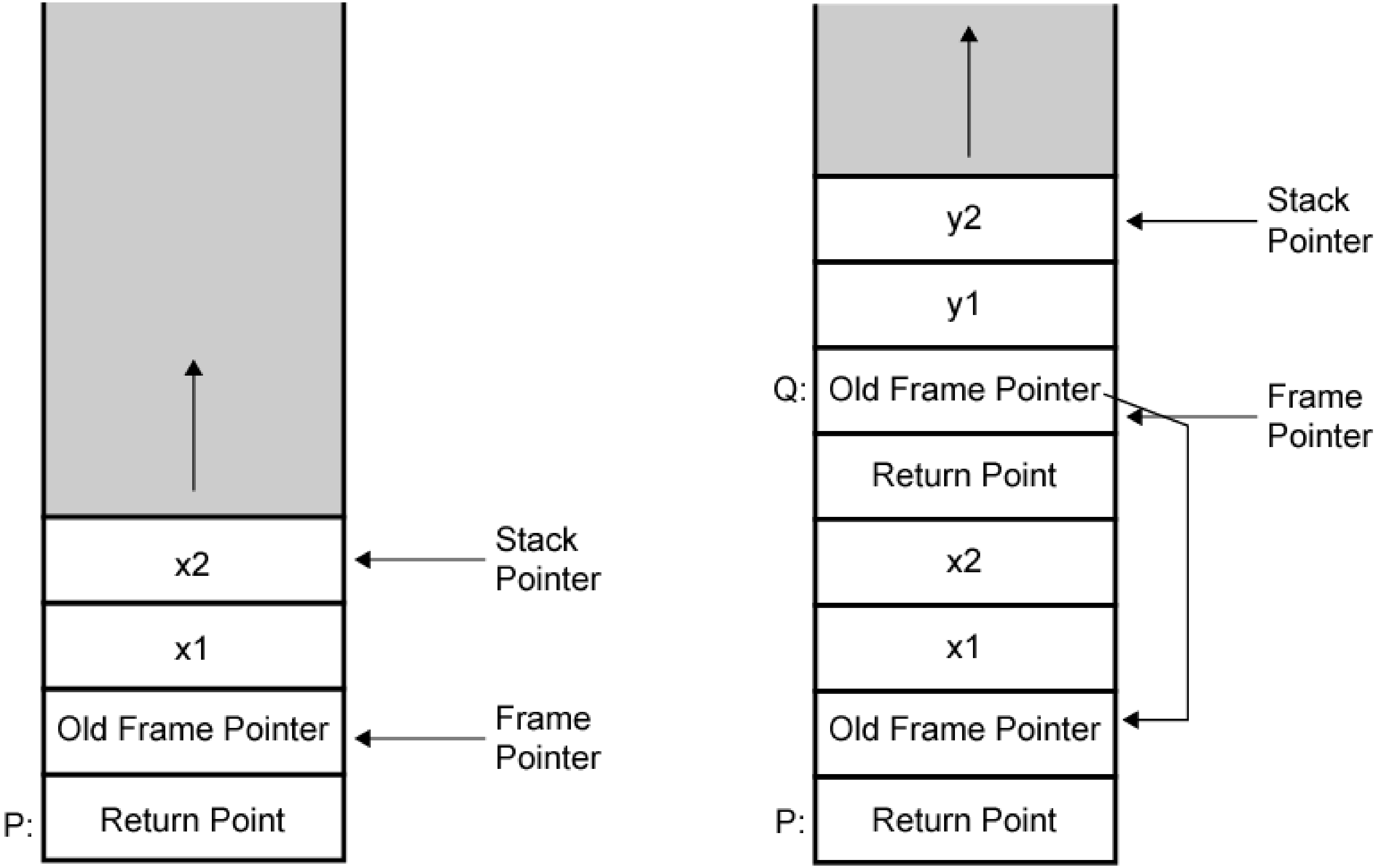
(a) Calls and returns

(b) Execution sequence

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

Stack Frame Growth Using Sample Procedures P and Q



(a) P is active

(b) P has called Q

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

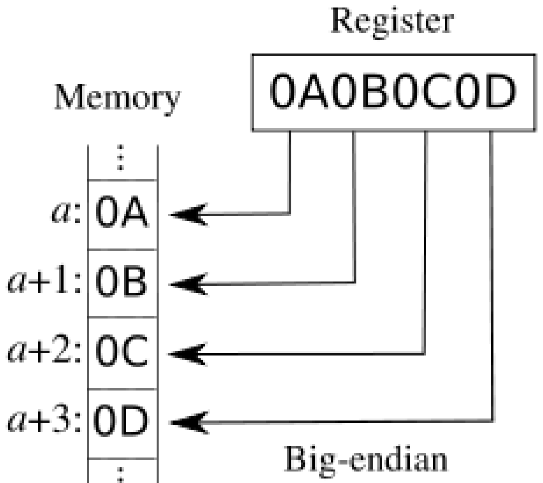
Byte Order (example)

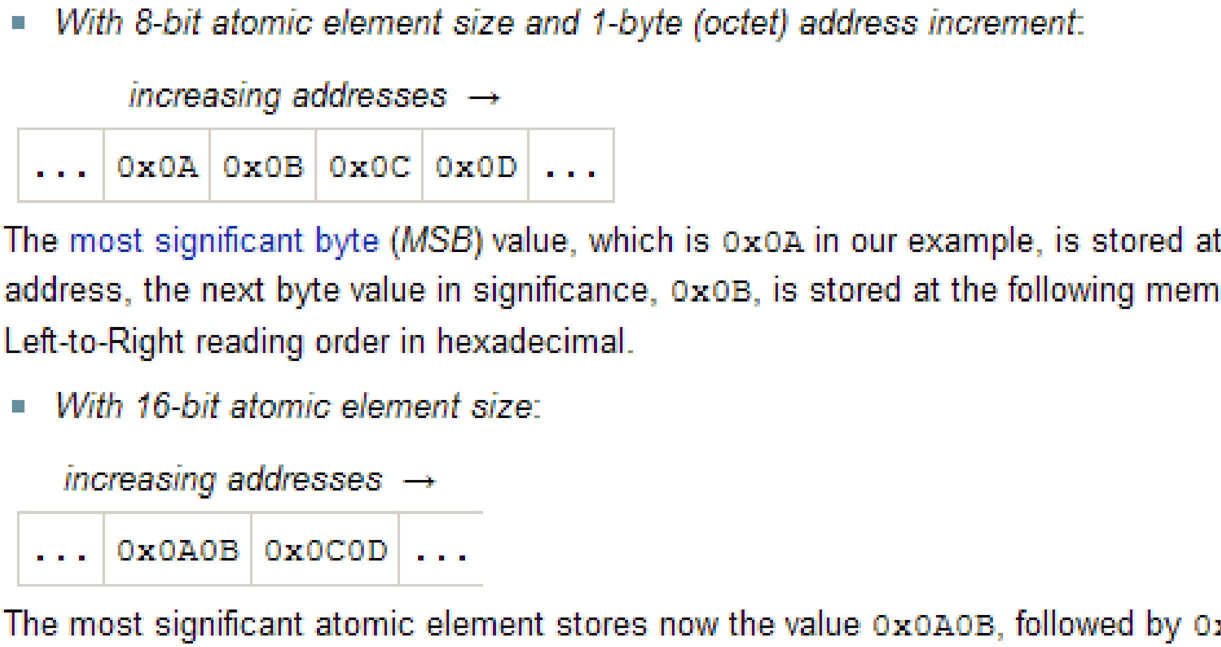
Address	Value (1)	Value(2)
184	12	78
185	34	56
186	56	34
186	78	12

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

Byte Order Names

- The problem is called Endian
- Big-endian





- *With 8-bit atomic element size and 1-byte (octet) address increment:*

increasing addresses →

...	0x0A	0x0B	0x0C	0x0D	...

The most significant byte (*MSB*) value, which is `0x0A` in our example, is stored at the memory location with the lowest address, the next byte value in significance, `0x0B`, is stored at the following memory location and so on. This is akin to Left-to-Right reading order in hexadecimal.

- *With 16-bit atomic element size:*

increasing addresses →

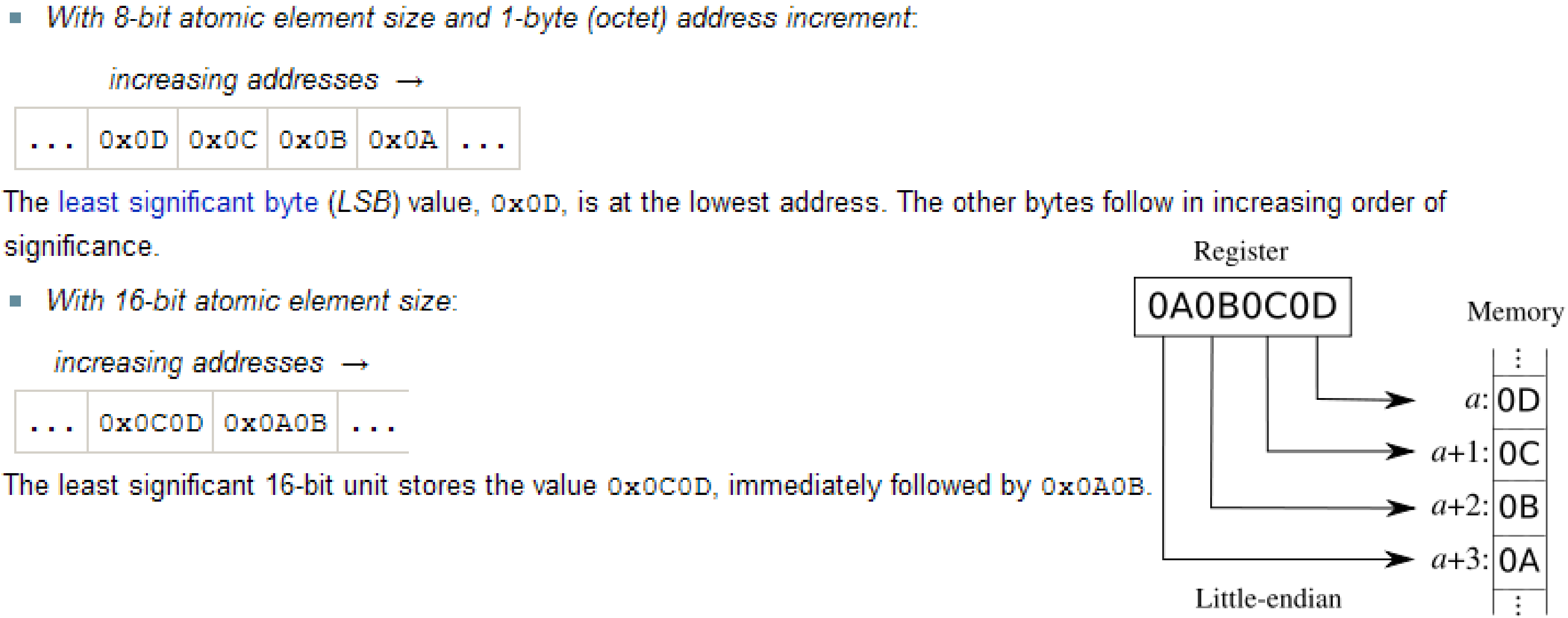
...	0x0A0B	0x0C0D	...

The most significant atomic element stores now the value `0x0A0B`, followed by `0x0C0D`.

[source : wikipedia]

Byte Order Names

- Little-endian



- *With 8-bit atomic element size and 1-byte (octet) address increment:*

increasing addresses →

...	`0x0D`	`0x0C`	`0x0B`	`0x0A`	...

The least significant byte (*LSB*) value, `0x0D`, is at the lowest address. The other bytes follow in increasing order of significance.

- *With 16-bit atomic element size:*

increasing addresses →

...	`0x0C0D`	`0x0A0B`	...

The least significant 16-bit unit stores the value `0x0C0D`, immediately followed by `0x0A0B`.

[source : wikipedia]

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;               // 0x6162_6364
```

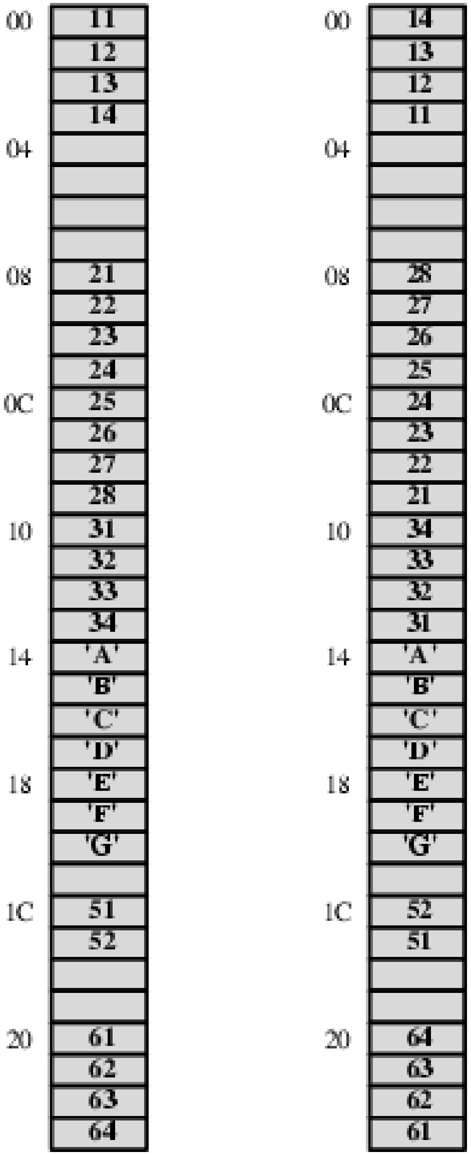
Big-endian address mapping

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
07	06	05	04	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
1F	1E	51 1D	52 1C	1B	'G' 1A	'F' 19	'E' 18	18
				61 23	62 22	63 21	64 20	20

Alternative View of Memory Map



00	11	14
	12	13
	13	12
	14	11
04		
08	21	28
	22	27
	23	26
	24	25
0C	25	24
	26	23
	27	22
	28	21
10	31	34
	32	33
	33	32
	34	31
14	'A'	'A'
	'B'	'B'
	'C'	'C'
	'D'	'D'
18	'E'	'E'
	'F'	'F'
	'G'	'G'
1C	51	52
	52	51
20	61	64
	62	63
	63	62
	64	61

(a) Big-endian

(b) Little-endian

Standard...What Standard?

- Pentium (80x86), VAX are little-endian
- IBM 370, Motorola 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