William Stallings
Computer Organization and Architecture
9th 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

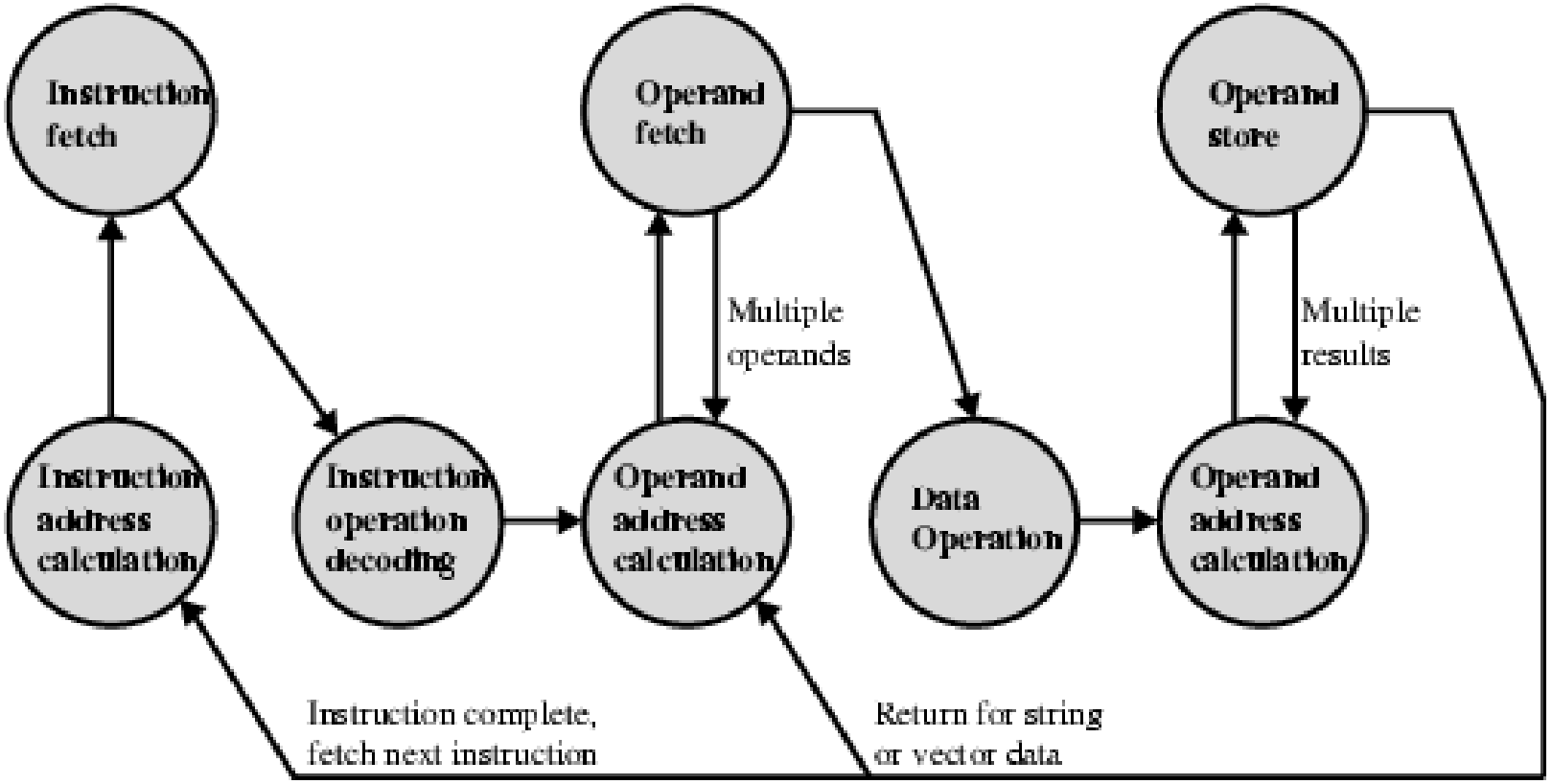
Elements of an Instruction

- 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?

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

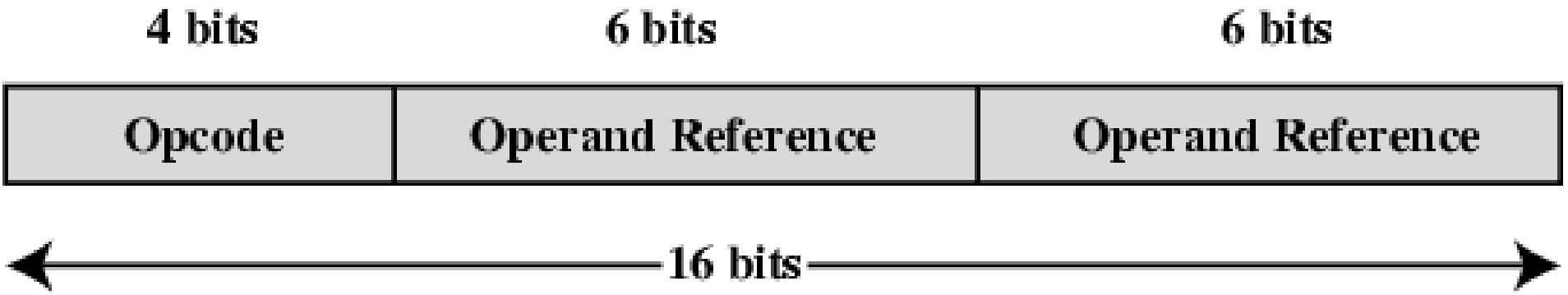
Instruction Cycle State Diagram



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 (arithmetic & logic inst.)
- Data storage (main memory, memory inst.)
- Data movement (I/O inst.)
- Program flow control

Number of Addresses

- One of the traditional ways of describing processor architecture is in terms of no. of addresses contained in each inst.
  - What is the max no. of addresses one might need in an inst.?
  - Arithmetic & logic inst. will require the most operands
  - All arithmetic & logical op are either one or two operands, thus we would need a max of 2 addresses to reference operands
  - Result of an op must be stored, suggesting a third address, finally, after completion of an inst., next inst. must be fetched & its address is needed (concluding)
  - An inst. could possibly be required to contain 4-address references: two op, one result & address of next inst. (in practice 4-address inst. are extremely rare)

Number of Addresses (a)

- **3 addresses**
  - —Operand 1, Operand 2, Result
  - —a = b + c;
  - —May be a forth - next instruction (usually implicit)
  - —3-address inst. formats are not common, because they require a relatively long inst. Format to hold three address references
  - —Needs very long words to hold everything
  - —Compare (1,2,3 address inst.) to compute

    Y = (A-B) / (C+D*E)

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

    e.g., LOAD B (AC ← B), implicit AC address

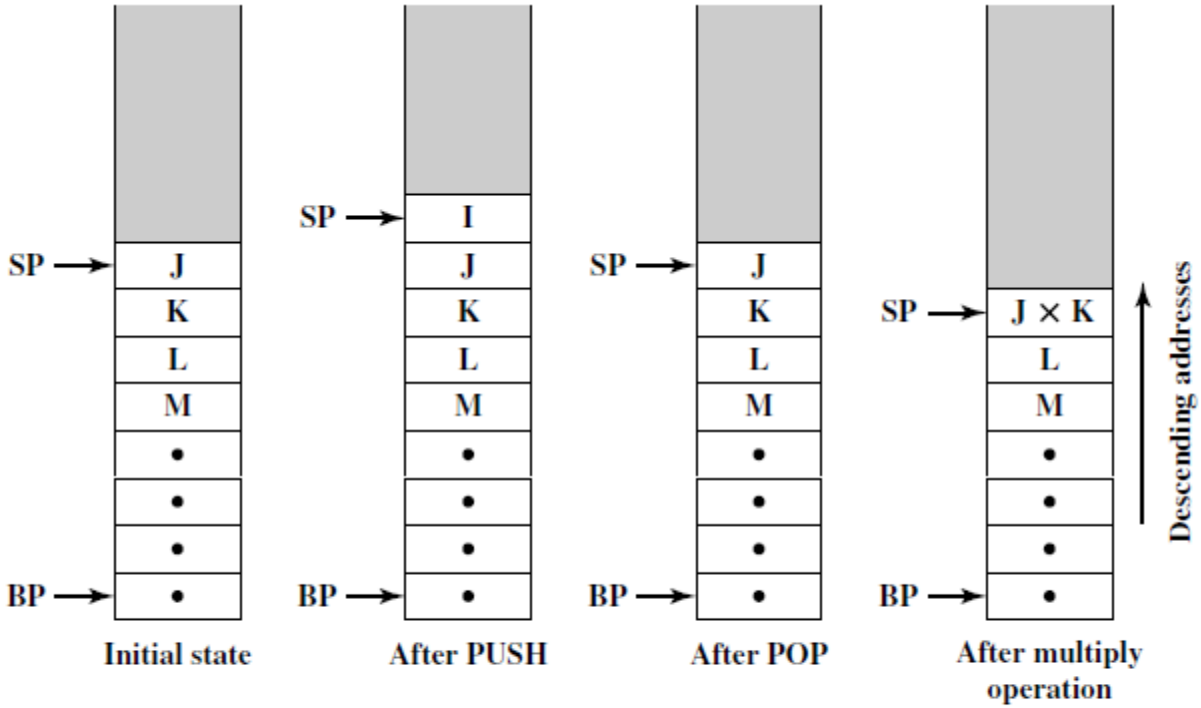
Number of Addresses (d)

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

  - —c = a + b

Number of Addresses (d)

- **0 (zero) addresses**



SP = Stack pointer
BP = Base pointer

Figure 10.13 Basic Stack Operation (full/descending)

Number of Addresses (Example)

Instruction		Comment
SUB	Y, A, B	$Y \leftarrow A - B$
MPY	T, D, E	$T \leftarrow D \times E$
ADD	T, T, C	$T \leftarrow T + C$
DIV	Y, Y, T	$Y \leftarrow Y \div T$

(a) Three-address instructions

Instruction		Comment
MOVE	Y, A	$Y \leftarrow A$
SUB	Y, B	$Y \leftarrow Y - B$
MOVE	T, D	$T \leftarrow D$
MPY	T, E	$T \leftarrow T \times E$
ADD	T, C	$T \leftarrow T + C$
DIV	Y, T	$Y \leftarrow Y \div T$

(b) Two-address instructions

Instruction		Comment
LOAD	D	$AC \leftarrow D$
MPY	E	$AC \leftarrow AC \times E$
ADD	C	$AC \leftarrow AC + C$
STOR	Y	$Y \leftarrow AC$
LOAD	A	$AC \leftarrow A$
SUB	B	$AC \leftarrow AC - B$
DIV	Y	$AC \leftarrow AC \div Y$
STOR	Y	$Y \leftarrow AC$

(c) One-address instructions

Figure 10.3 Programs to Execute $Y = \dfrac{A - B}{C + (D \times E)}$

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

How Many Addresses

- Compares typical one, two, three addresses instructions:
- Programs to Execute e.g.,

$$Y = (A - B) / (C + D * E)$$

Design Decisions (1)

- Operation-Wise
  - 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… ch: 11)
- RISC v CISC (later… ch: 13)

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!)

Example- Pentium & Power PC Data Types

- 8 bit Byte
- 16 bit word
- 32 bit double word
- 64 bit quad word
- Addressing is by 8 bit unit

Types of Operation

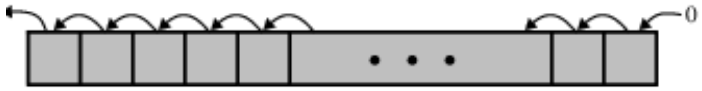
- Data Transfer
- Arithmetic
- Logical
- Conversion
- I/O
- System Control
- Transfer of Control

Arithmetic

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

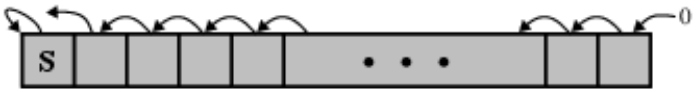
Shift and Rotate Operations

(a) Logical right shift

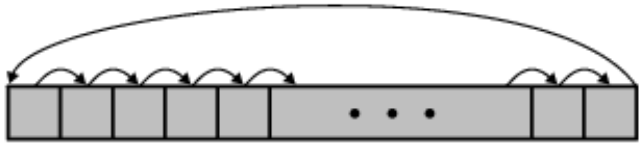


(b) Logical left shift

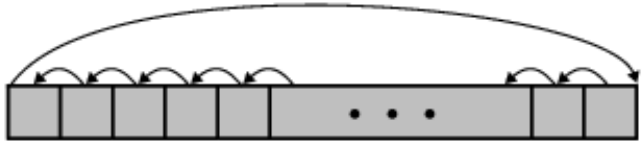
(c) Arithmetic right shift



(d) Arithmetic left shift



(e) Right rotate



(f) Left rotate

Logical

- Bitwise operations
- AND, OR, NOT

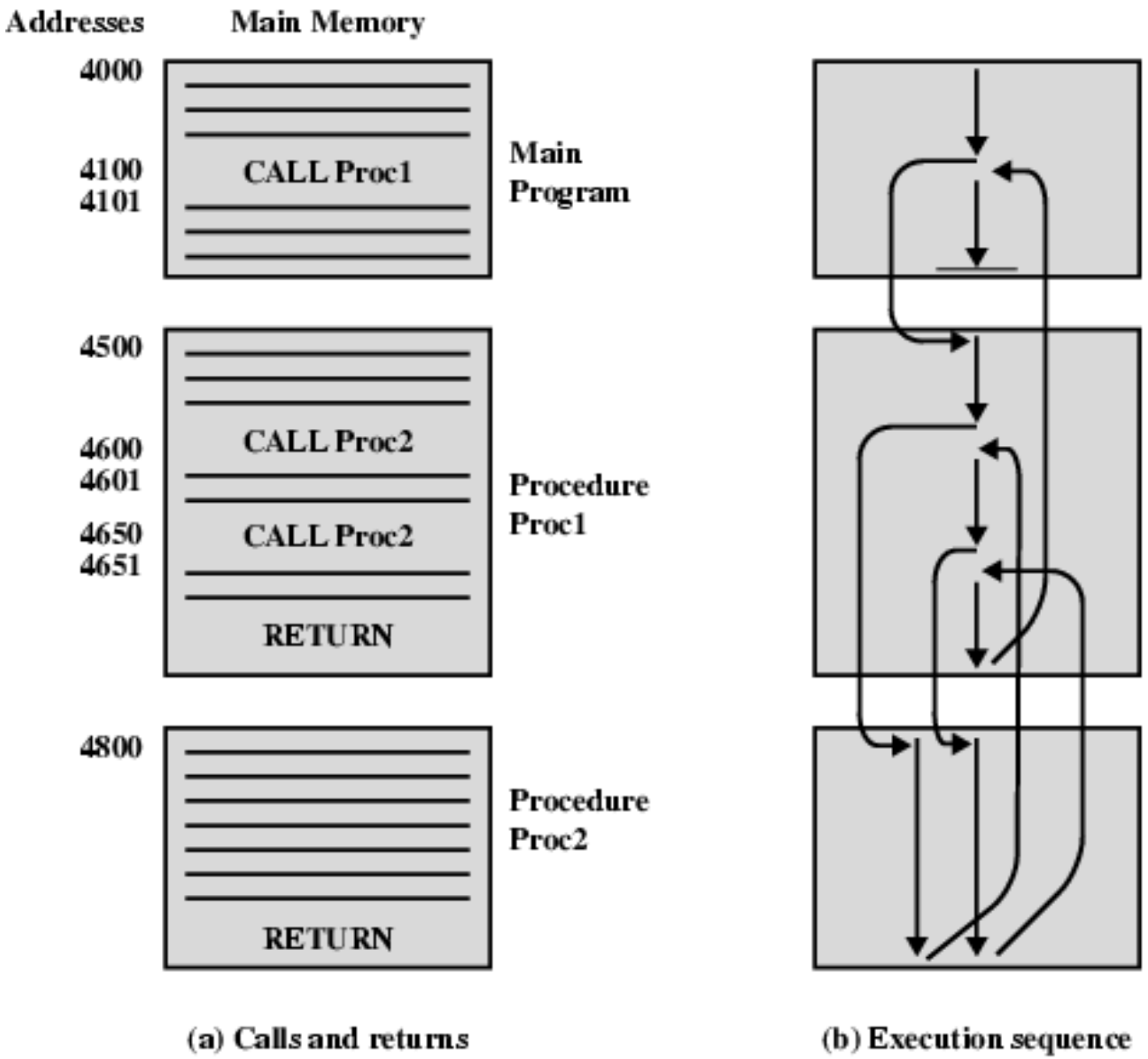
Conversion

- E.g. Binary to Decimal

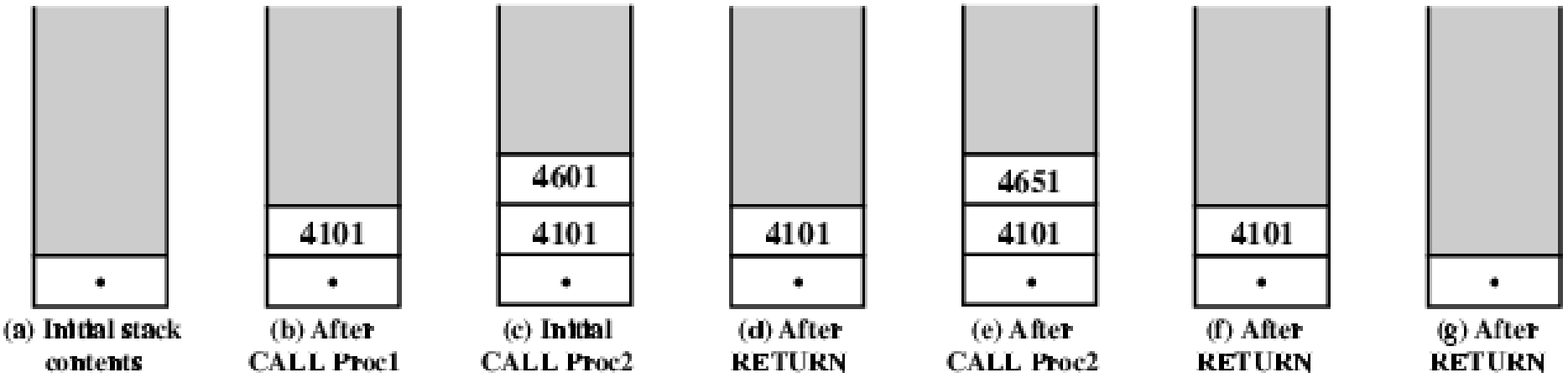
Transfer of Control

- 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

Nested Procedure Calls



Use of Stack


4601
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

Exercise For Reader

- Find out about instruction set for Pentium and PowerPC
- 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;
```

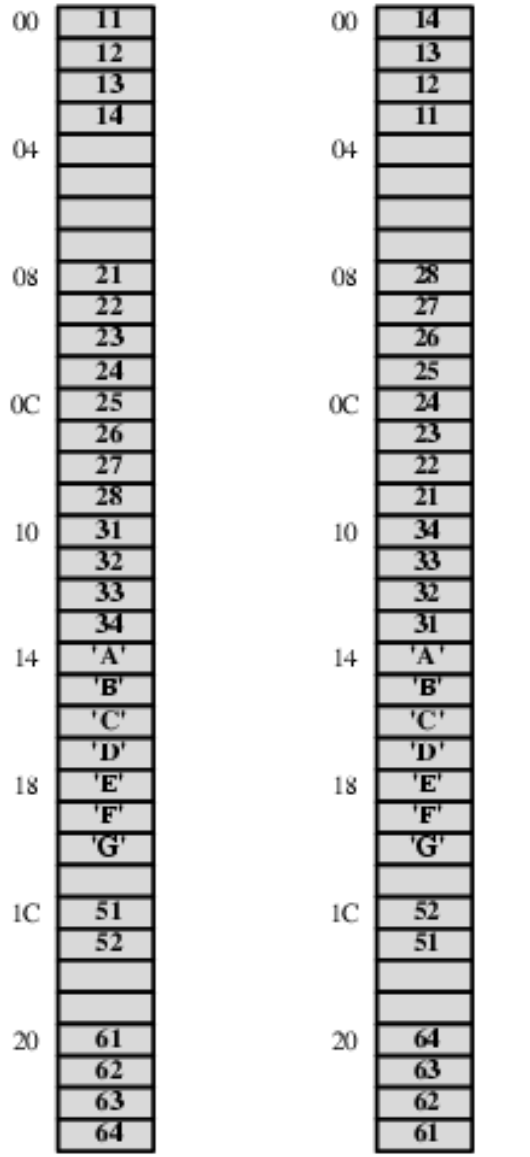
Big-endian address mapping

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



(a) Big-endian (b) Little-endian

Question…

- Assume a stack-oriented processor that includes the stack operations ***PUSH*** and ***POP***. Arithmetic operations automatically involve the top one or two stack elements. Begin with an empty stack. What stack elements remain after the following instructions are executed?

- **PUSH 4**
- **PUSH 7**
- **PUSH 8**
- **ADD**
- **PUSH 10**
- **SUB**
- **MUL**
- **PUSH 9**
- **ADD**