

CS2100

<http://www.comp.nus.edu.sg/~cs2100/>

COMPUTER ORGANISATION

## Lecture #10

---

# Instruction Set Architecture (ISA)



**NUS**  
National University  
of Singapore

School of  
Computing

# Lecture #10: Instruction Set Architecture

1. Overview
2. RISC vs CISC: The Famous Battle
3. The 5 Concepts in ISA Design
  - 3.1 Concept #1: Data Storage
  - 3.2 Concept #2: Memory and Addressing Mode
  - 3.3 Concept #3: Operations in Instruction Set
  - 3.4 Concept #4: Instruction Formats
  - 3.5 Concept #5: Encoding the Instruction Set

# 1. Overview

- We have studied MIPS but it is only one example
  - There are many other assembly languages with different characteristics
- This lecture gives a more **general view** on the design of Instruction Set Architecture (ISA)
- Use your understanding of MIPS and explore other possibilities/alternatives

## 2. RISC vs CISC: The Famous Battle

- Two major design philosophies for ISA:
- **Complex Instruction Set Computer (CISC)**
  - Example: x86-32 (IA32)
  - Single instruction performs complex operation
    - VAX architecture had an instruction to multiply polynomials
  - Smaller program size as memory was premium
  - Complex implementation, no room for hardware optimization
- **Reduced Instruction Set Computer (RISC)**
  - Example: **MIPS**, ARM
  - Keep the instruction set small and simple, makes it easier to build/optimize hardware
  - Burden on software to combine simpler operations to implement high-level language statements

# 3. The 5 Concepts in ISA Design

1. Data Storage

2. Memory Addressing Modes

3. Operations in the Instruction Set

4. Instruction Formats

5. Encoding the Instruction Set

## 3.1 Concept #1: Data Storage

- Storage Architecture
- General Purpose Register Architecture

### **Concept #1: Data Storage**

Concept #2: Memory Addressing Modes

Concept #3: Operations in the Instruction Set

Concept #4: Instruction Formats

Concept #5: Encoding the Instruction Set

## 3.1 Storage Architecture: Definition



- von Neumann Architecture:
  - Data (operands) are stored in memory
- For a processor, **storage architecture** concerns with:
  - Where do we store the operands so that the computation can be performed?
  - Where do we store the computation result afterwards?
  - How do we specify the operands?
- Major storage architectures ... (next slide)

## 3.1 Storage Architecture: Common Design

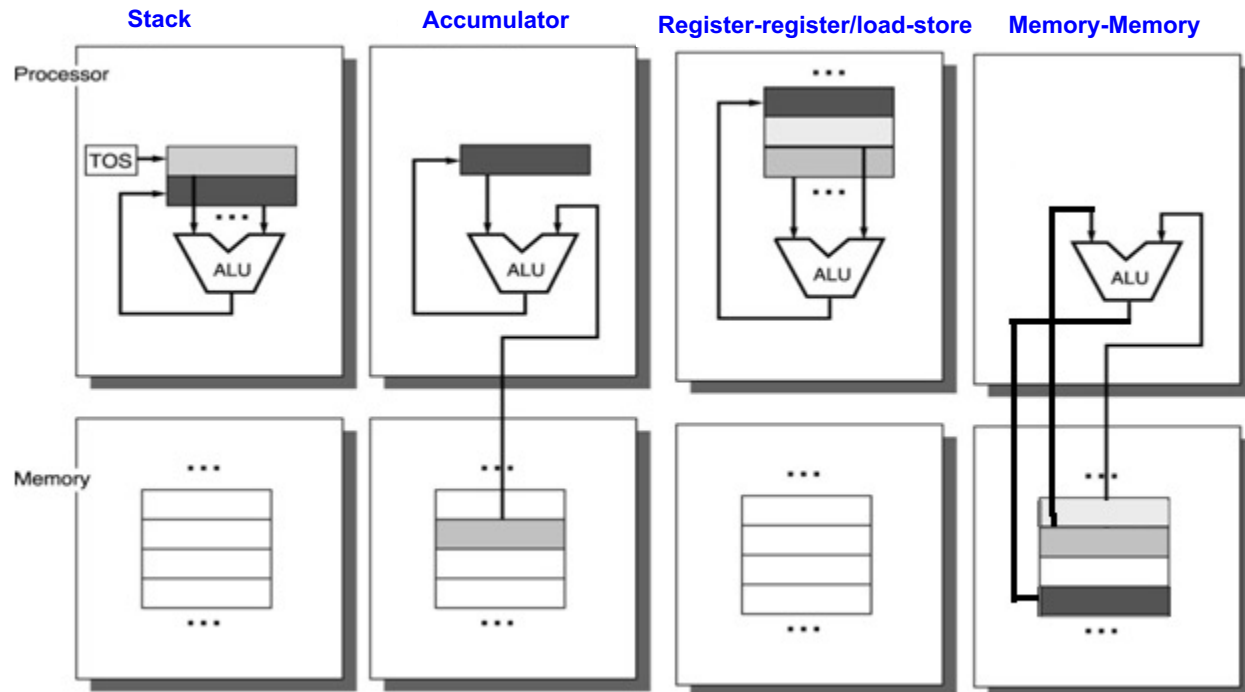
- **Stack architecture:**
  - Operands are implicitly on top of the stack.
- **Accumulator architecture:**
  - One operand is implicitly in the accumulator (a special register).  
Examples: IBM 701, DEC PDP-8.
- **General-purpose register architecture:**
  - Only explicit operands.
  - **Register-memory architecture** (one operand in memory).  
Examples: Motorola 68000, Intel 80386.
  - **Register-register (or load-store) architecture.**  
Examples: MIPS, DEC Alpha.
- **Memory-memory architecture:**
  - All operands in memory. Example: DEC VAX.



# 3.1 Storage Architecture: Example

Stack	Accumulator	Register (load-store)	Memory-Memory
Push A	Load A	Load R1, A	Add C, A, B
Push B	Add B	Load R2, B	
Add	Store C	Add R3, R1, R2	
Pop C		Store R3, C	

$$C = A + B$$



## 3.1 Storage Architecture: GPR Architecture

- For modern processors:
  - **General-Purpose Register** (GPR) is the most common choice for storage design
  - **RISC** computers typically uses **Register-Register (Load/Store)** design
    - E.g. MIPS, ARM
  - **CISC** computers use a mixture of Register-Register and Register-Memory
    - E.g. IA32

## 3.2 Concept #2: Memory Addressing Mode

- Memory Locations and Addresses
- Addressing Modes

Concept #1: Data Storage

**Concept #2: Memory Addressing Modes**

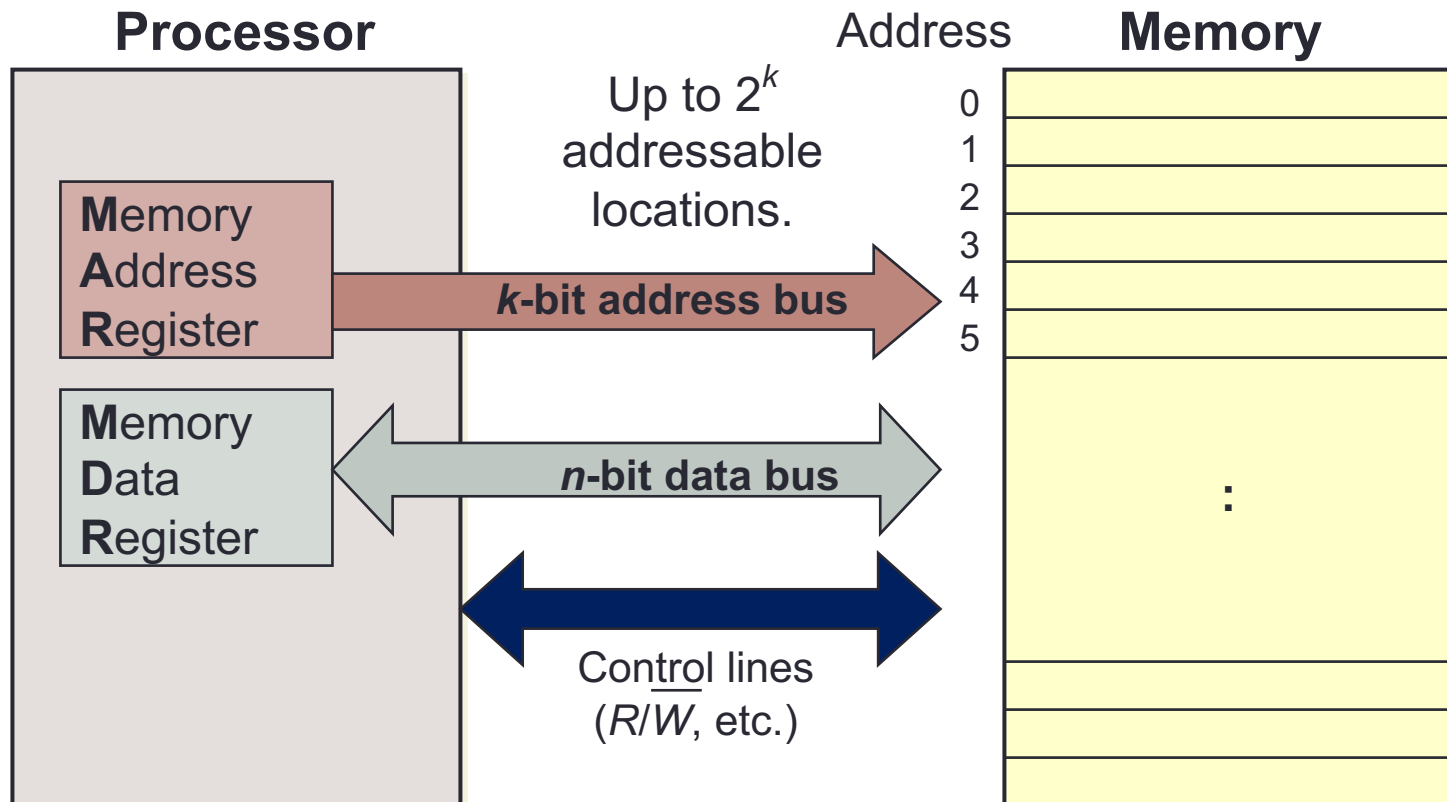
Concept #3: Operations in the Instruction Set

Concept #4: Instruction Formats

Concept #5: Encoding the Instruction Set

## 3.2 Memory Address and Content

- Given  $k$ -bit address, the address space is of size  $2^k$
- Each memory transfer consists of one word of  $n$  bits



## 3.2 Memory Content: Endianness

### ■ Endianness:

- The relative ordering of the bytes in a multiple-byte word stored in memory

#### Big-endian:

Most significant byte stored in lowest address.

Example:

IBM 360/370, Motorola 68000, MIPS (Silicon Graphics), SPARC.

Example: **0xDE AD BE EF**

Stored as:

0	DE
1	AD
2	BE
3	EF

#### Little-endian:

Least significant byte stored in lowest address.

Example:

Intel 80x86, DEC VAX, DEC Alpha.

Example: **0xDE AD BE EF**

Stored as:

0	EF
1	BE
2	AD
3	DE

## 3.2 Addressing Modes

- Addressing Mode:
  - Ways to specify an operand in an assembly language
- In MIPS, there are only 3 addressing modes:
  - **Register:**
    - Operand is in a register (eg: `add $t1, $t2, $t3`)
  - **Immediate:**
    - Operand is specified in the instruction directly (eg: `addi $t1, $t2, 98`)
  - **Displacement:**
    - Operand is in memory with address calculated as `Base + Offset` (eg: `lw $t1, 20($t2)`)

## 3.2 Addressing Modes: Others

<u>Addressing mode</u>	<u>Example</u>	<u>Meaning</u>
<b>Register</b>	Add R4,R3	$R4 \leftarrow R4 + R3$
<b>Immediate</b>	Add R4,#3	$R4 \leftarrow R4 + 3$
<b>Displacement</b>	Add R4,100(R1)	$R4 \leftarrow R4 + \text{Mem}[100 + R1]$
<b>Register indirect</b>	Add R4,(R1)	$R4 \leftarrow R4 + \text{Mem}[R1]$
<b>Indexed / Base</b>	Add R3,(R1+R2)	$R3 \leftarrow R3 + \text{Mem}[R1 + R2]$
<b>Direct or absolute</b>	Add R1,(1001)	$R1 \leftarrow R1 + \text{Mem}[1001]$
<b>Memory indirect</b>	Add R1,@(R3)	$R1 \leftarrow R1 + \text{Mem}[\text{Mem}[R3]]$
<b>Auto-increment</b>	Add R1,(R2)+	$R1 \leftarrow R1 + \text{Mem}[R2]; R2 \leftarrow R2 + d$
<b>Auto-decrement</b>	Add R1,-(R2)	$R2 \leftarrow R2 - d; R1 \leftarrow R1 + \text{Mem}[R2]$
<b>Scaled</b>	Add R1,100(R2)[R3]	$R1 \leftarrow R1 + \text{Mem}[100 + R2 + R3 * d]$

## 3.3 Concept #3: Operations in Instructions Set

- Standard Operations in an Instruction Set
- Frequently Used Instructions

Concept #1: Data Storage

Concept #2: Memory Addressing Modes

**Concept #3: Operations in the Instruction Set**

Concept #4: Instruction Formats

Concept #5: Encoding the Instruction Set



## 3.3 Standard Operations

### 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

integer (binary + decimal) or FP  
add, subtract, multiply, divide

### Shift

shift left/right, rotate left/right

### Logical

not, and, or, set, clear

---

### Control flow

Jump (unconditional), Branch (conditional)

### Subroutine Linkage

call, return

---

### Interrupt

trap, return

---

### Synchronization

test & set (atomic r-m-w)

### String

search, move, compare

### Graphics

pixel and vertex operations,  
compression/decompression

## 3.3 Frequently Used Instructions

Rank	Integer Instructions	Average %
1	Load	22%
2	Conditional Branch	20%
3	Compare	16%
4	Store	12%
5	Add	8%
6	Bitwise AND	6%
7	Sub	5%
8	Move register to register	4%
9	Procedure call	1%
10	Return	1%
	<b>Total</b>	<b>96%</b>

Make these instructions fast!  
Amdahl's law – make the  
common cases fast!

## 3.4 Concept #4: Instruction Formats

- Instruction Length
- Instruction Fields
  - Type and Size of Operands

Concept #1: Data Storage

Concept #2: Memory Addressing Modes

Concept #3: Operations in the Instruction Set

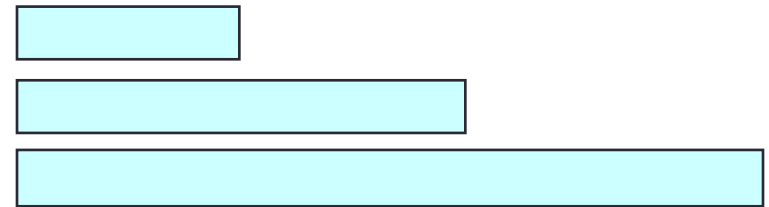
**Concept #4: Instruction Formats**

Concept #5: Encoding the Instruction Set

## 3.4 Instruction Length

- **Variable-length** instructions.

- Intel 80x86: Instructions vary from 1 to 17 bytes long.
- Digital VAX: Instructions vary from 1 to 54 bytes long.
- Require multi-step fetch and decode.
- Allow for a more flexible (but complex) and compact instruction set.



- **Fixed-length** instructions.

- Used in most RISC (Reduced Instruction Set Computers)
- MIPS, PowerPC: Instructions are 4 bytes long.
- Allow for easy fetch and decode.
- Simplify pipelining and parallelism.
- Instruction bits are scarce.

- **Hybrid** instructions: a mix of variable- and fixed-length instructions.

## 3.4 Instruction Fields

- An instruction consists of
  - **opcode**: unique code to specify the desired operation
  - **operands**: zero or more additional information needed for the operation
- The operation designates the type and size of the operands
  - **Typical type and size**: Character (8 bits), half-word (eg: 16 bits), word (eg: 32 bits), single-precision floating point (eg: 1 word), double-precision floating point (eg: 2 words).
- Expectations from any new 32-bit architecture:
  - Support for 8-, 16- and 32-bit integer and 32-bit and 64-bit floating point operations. A 64-bit architecture would need to support 64-bit integers as well.

## 3.5 Concept #5: Encoding the Instruction Set

- Instruction Encoding
- Encoding for Fixed-Length Instructions

Concept #1: Data Storage

Concept #2: Memory Addressing Modes

Concept #3: Operations in the Instruction Set

Concept #4: Instruction Formats

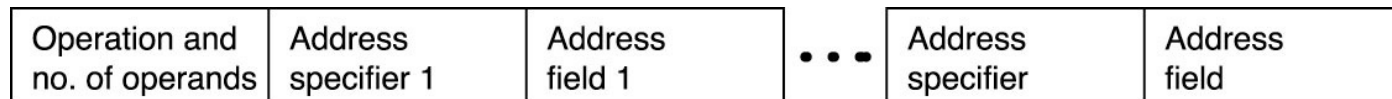
**Concept #5: Encoding the Instruction Set**

## 3.5 Instruction Encoding: Overview

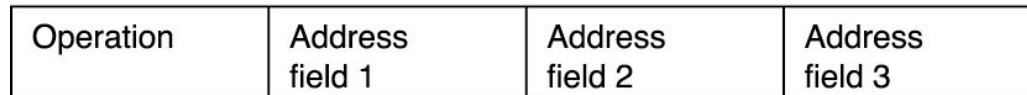
- How are instructions represented in binary format for execution by the processor?
- Issues:
  - Code size, speed/performance, design complexity.
- Things to be decided:
  - Number of registers
  - Number of addressing modes
  - Number of operands in an instruction
- The different competing forces:
  - Have many registers and addressing modes
  - Reduce code size
  - Have instruction length that is easy to handle (fixed-length instructions are easier to handle)

## 3.5 Encoding Choices

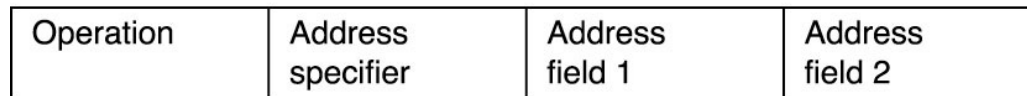
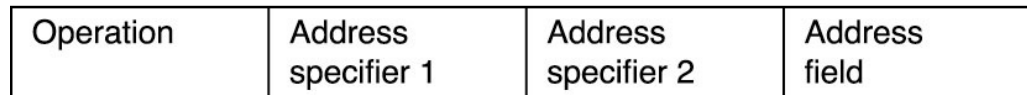
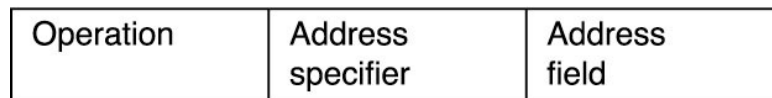
- Three encoding choices: variable, fixed, hybrid.



(a) Variable (e.g., VAX, Intel 80x86)



(b) Fixed (e.g., Alpha, ARM, MIPS, PowerPC, SPARC, SuperH)



(c) Hybrid (e.g., IBM 360/70, MIPS16, Thumb, TI TMS320C54x)



## 3.5 Fixed Length Instructions: Encoding (1/4)

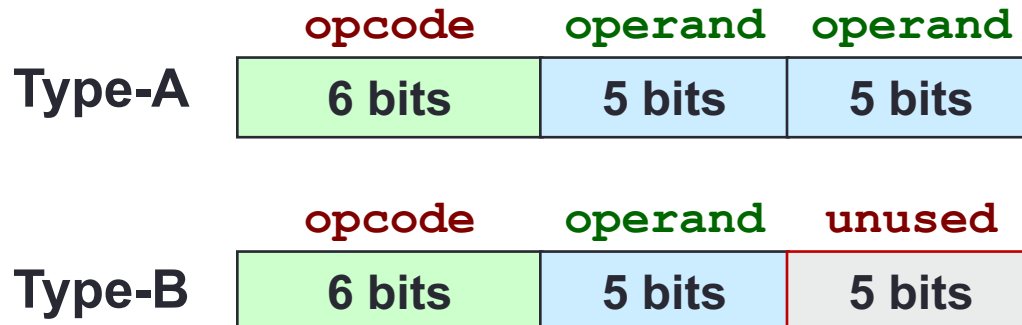
- Fixed length instruction presents a much more interesting challenge:
  - Q: How to fit multiple sets of instruction types into same (limited) number of bits?
  - A: Work with the most constrained instruction types first
- **Expanding Opcode** scheme:
  - The opcode has variable lengths for different instructions.
  - A good way to maximize the instruction bits.

## 3.5 Fixed Length Instructions: Encoding (2/4)

### ■ Example:

- 16-bit fixed length instructions, with 2 types of instructions
- **Type-A:** 2 operands, each operand is 5-bit
- **Type-B:** 1 operand of 5-bit

**First Attempt:**  
Fixed length Opcode



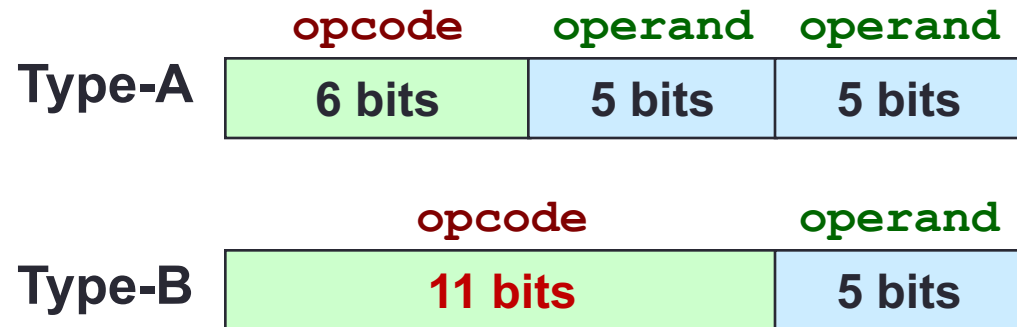
### Problem:

- Wasted bits in Type-B instructions
- Maximum total number of instructions is  $2^6$  or 64.

## 3.5 Fixed Length Instructions: Encoding (3/4)

- Use **expanding opcode** scheme:
  - Extend the opcode for type-B instructions to **11 bits**
- ➔ No wasted bits and result in a larger instruction set

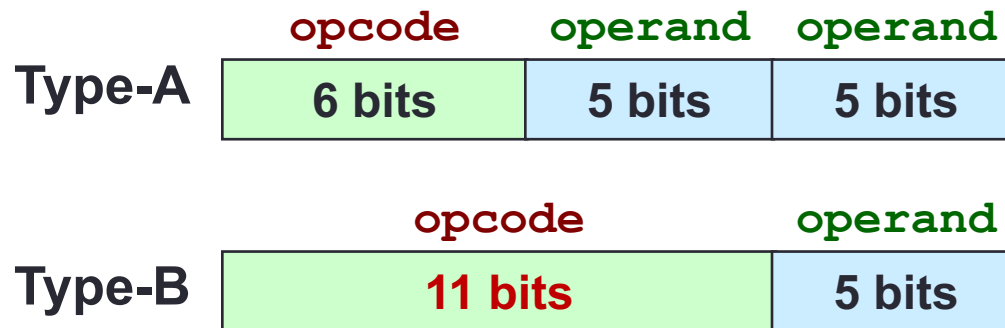
**Second Attempt:**  
Expanding Opcode



- **Questions:**
  - How do we distinguish between Type-A and Type-B?
  - How many different instructions do we really have?

## 3.5 Fixed Length Instructions: Encoding (4/4)

- What is the maximum number of instructions?



**Answer:**

$$\begin{aligned}
 &1 + (2^6 - 1) \times 2^5 \\
 &= 1 + 63 \times 32 \\
 &= 2017
 \end{aligned}$$

- Reasoning:**

- For every 6-bit prefix (front-part) given to Type-B, we get  $2^5$  unique patterns, e.g. [111111]xxxxxx
- So, we should minimize Type-A instruction and give **as many 6-bit prefixes as possible to Type-B**
  - 1 Type-A instruction,  $2^6 - 1$  prefixes for Type-B

## 3.5 Expanding Opcode: Another Example

- Design an expanding opcode for the following to be encoded in a 36-bit instruction format. An address takes up 15 bits and a register number 3 bits.
  - 7 instructions with two addresses and one register number.
  - 500 instructions with one address and one register number.
  - 50 instructions with no address or register.

One possible answer:

3 bits	15 bits	15 bits	3 bits
000 → 110 opcode	address	address	register
111 ←	000000 + 9 bits opcode →	address	register
111 ←	000001 : + 9 0s 110010 opcode →	unused	unused

# Past Midterm/Exam Questions (1/2)

- A certain machine has 12-bit instructions and 4-bit addresses. Some instructions have one address and others have two. Both types of instructions exist in the machine.

1. What is the maximum number of instructions with one address?

- a) 15
- b) 16
- c) 240
- d) 256
- e) None of the above

Answer: (c)

# Past Midterm/Exam Questions (2/2)

- A certain machine has 12-bit instructions and 4-bit addresses. Some instructions have one address and others have two. Both types of instructions exist in the machine.
2. What is the minimum total number of instructions, assuming the encoding space is completely utilized (that is, no more instructions can be accommodated)?
- a) 31
  - b) 32
  - c) 48
  - d) 256
  - e) None of the above
- Answer: (a)**

# Reading

- **Instructions: Language of the Computer**
  - COD Chapter 2, pg 46-53, 58-71. (3<sup>rd</sup> edition)
  - COD Chapter 2, pg 74-81, 86-87, 94-104. (4<sup>th</sup> edition)





End of File