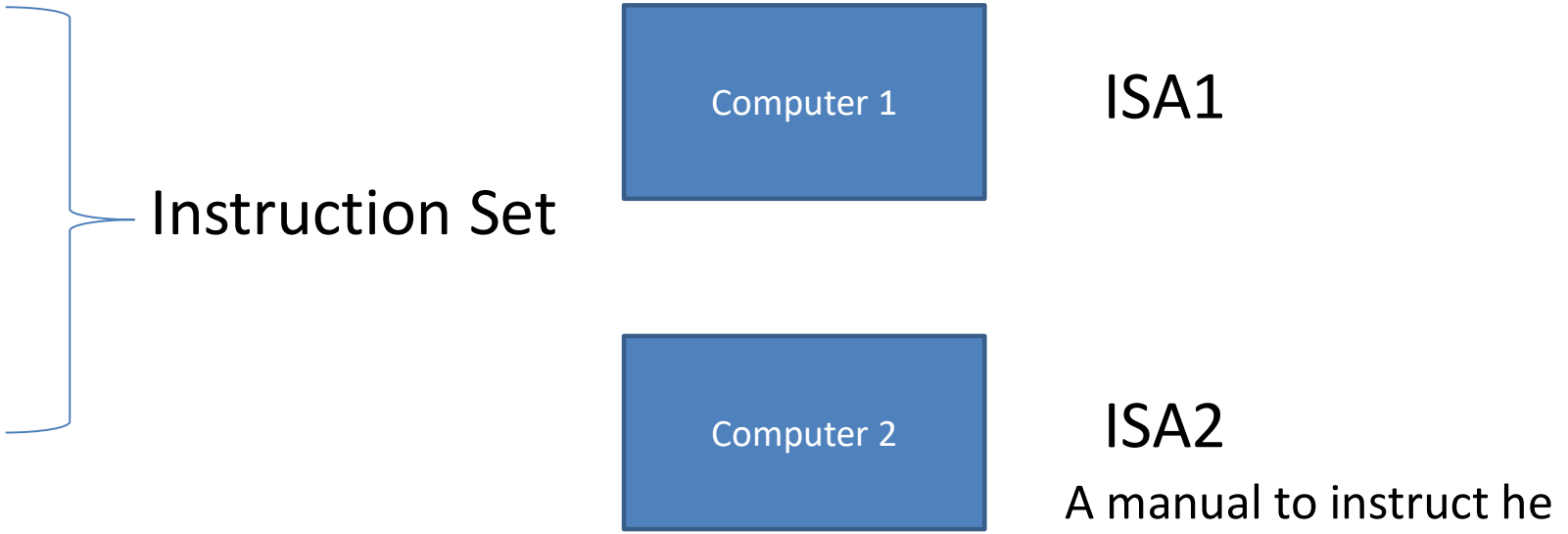
Chapter 2

Instructions: Language of the Computer

Review

Instruction Set

- Add
- Multiply
- Divide
- Load Data



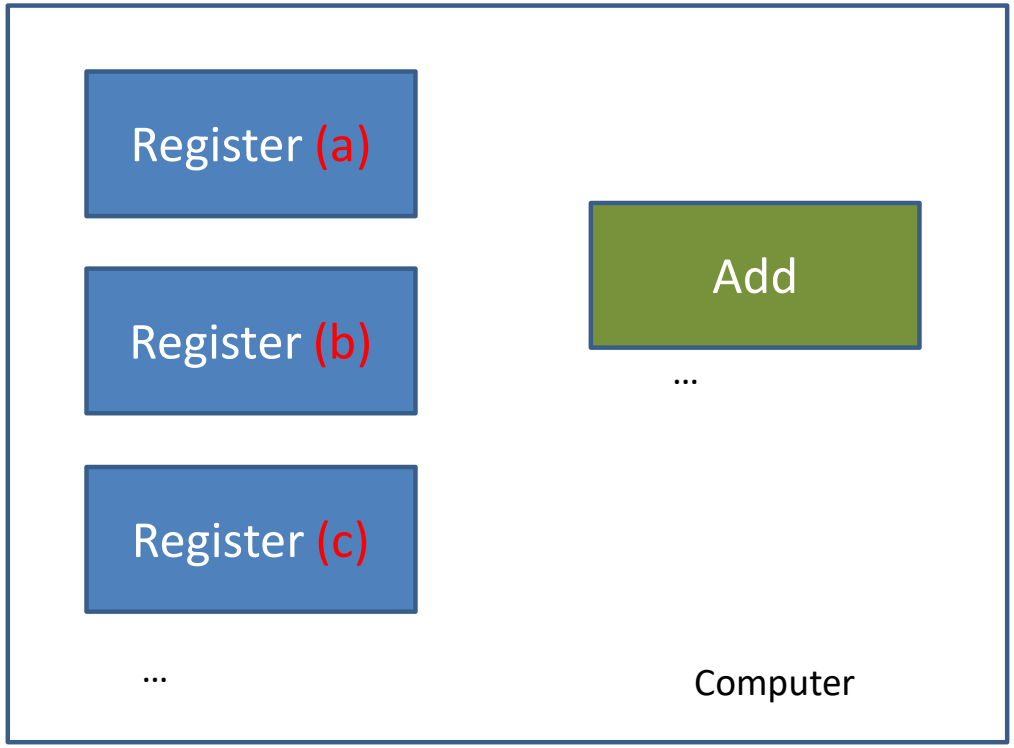
Review

The ARMv8 Instruction Set

- A subset, called LEGv8, used as the example throughout the book
- Commercialized by ARM Holdings (www.arm.com)
- Large share of embedded core market
  - Applications in consumer electronics, network/storage equipment, cameras, printers, …
- Typical of many modern ISAs
  - See ARM Reference Data tear-out card

Review

Operations of Computer Hardware



1. Has multiple registers, and logic gates to perform operations. E.g. add
2. Registers contain/store data.
3. Operators (like Add), can only access data in the registers.

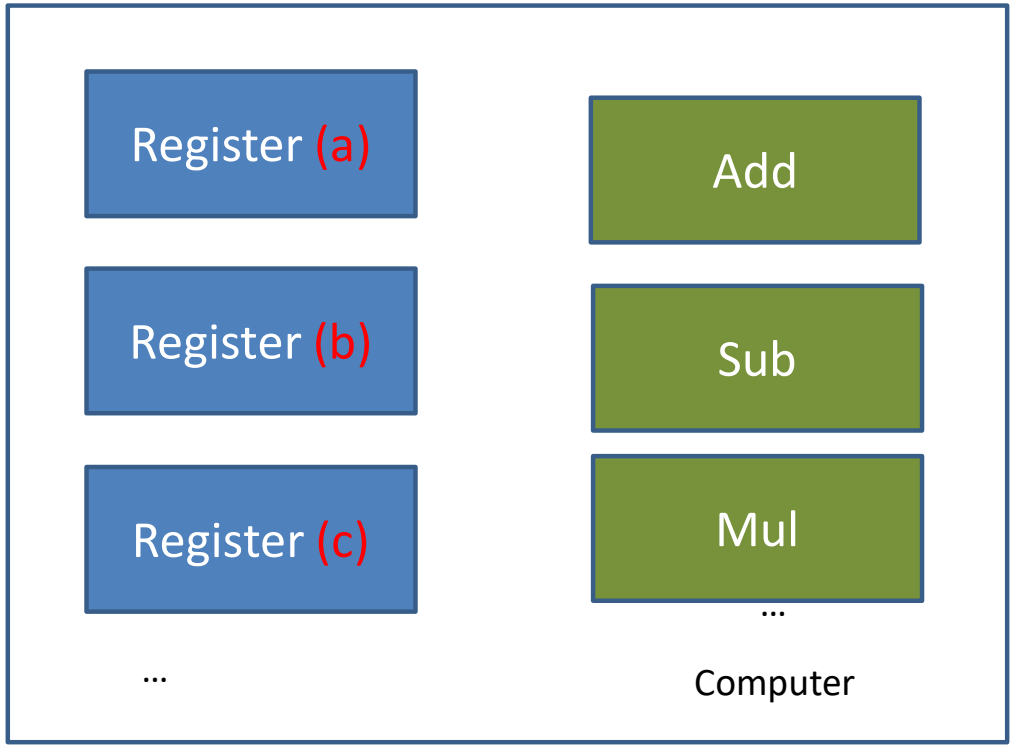
1. To instruct computer to
   1. **Add** (operation)
   2. **Values** in register **b and c** (Source Variables)
   3. Store the **result in a** (Destination Variable)

LEGv8 Instruction:

$$ADD\ a, b, c$$

Review

Operations of Computer Hardware



1. To instruct computer to
   1. **Add** (operation)
   2. **Values** in register **b and c** (Source Variables)
   3. Store the **result in a** (Destination Variable)

LEGv8 Instruction:

$$ADD\ a, b, c$$

One operation — Has three variables

Design Principle 1: Simplicity favors regularity

All LEGv8 **Arithmetic Instructions** perform only one operation and always has exactly three variables

$SUB\ a, b, c$ // subtract instrcution $(a = b - c)$
$MUL\ a, b, c$ // multiply instruction $(a = b * c)$

Review

Example - 1

$$a = b + c + d + e$$


```
ADD a, b, c   // a = b + c
ADD a, a, d   // a = a + d
ADD a, a, e   // a = a + e
```

3 instruction to sum 4 variables

Review

Example - 3

$$f = (g + h) - (i + j)$$

```
ADD t0, g, h   // t0 = g + h
ADD t1, i, j   // t1 = i + j
SUB f, t0, t1  // f = t0 - t1
```

t0, t1: temporary variables created by the compiler

Review

Example - 3

$$f = (g + h) - (i + j)$$

```
ADD t0, g, h   // t0 = g + h
ADD t1, i, j   // t1 = i + j
SUB f, t0, t1  // f = t0 - t1
```

Variables t0, t1, f, g, h, i, j

Stored in registers

Review

Bits, Bytes, Words, and Double Words

For our course!!!

0 ➔ 1 **bit** of data
1 ➔ 1 **bit** of data

10011101 (8 bits) ➔ 1 **byte** of data

Overtime this became a basic unit of data.
Older system represented letters using bytes
As a results most memory hardware

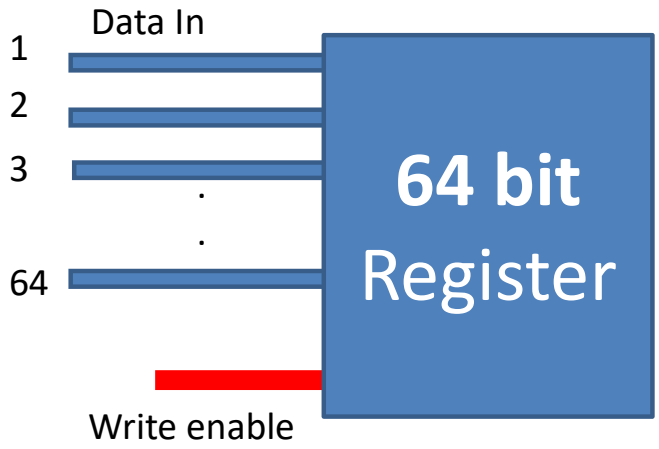
10011101 **1 byte** 10010001 **2 byte** 10010101 **3 byte** 10010101 **4 byte** ➔ **4 bytes** is a **word** (32 bits)

10011101 **1 byte** 10010001 **2 byte** 10010101 **3 byte** … 10010101 **7 byte** ➔ **8 bytes** is a **Doubleword** (64 bits)

Review

Operands of the Computer Hardware

- LEGv8 Register size – **64 Bits**
  - **Double words**



Review

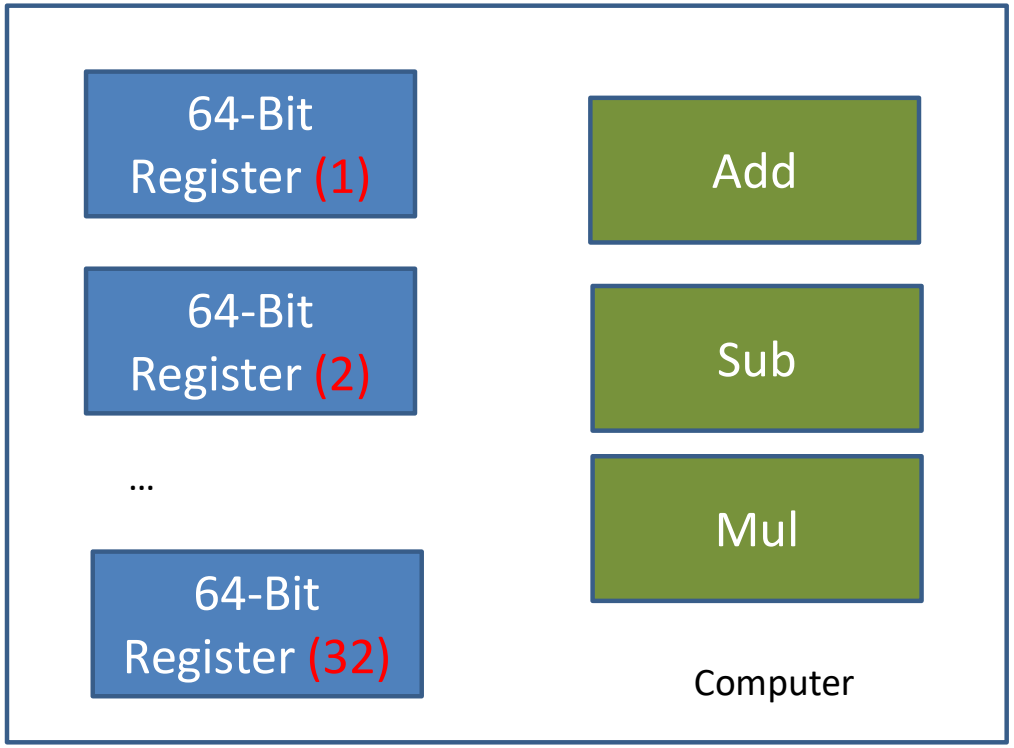
Operands of the Computer Hardware

- LEGv8 Register size – 64 Bits
- Total of **32 registers** (64-bit)

Why only 32??

Design Principle 2: Smaller is faster

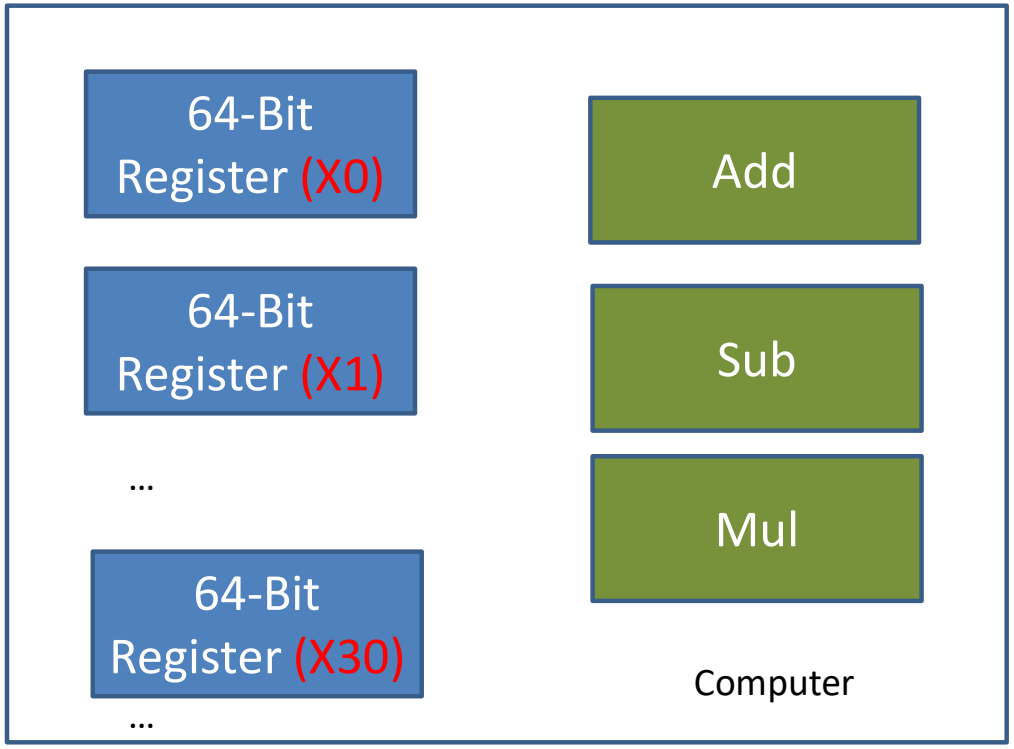
1. Having more registers may increase the clock cycle time (longer for electronic signals to travel)
2. Size of instructions (number of bits) is predefined and same for all instructions. More register requires more bits to specify registers.
   1. 32 registers – require 5 bits max
   2. 64 registers may require 6 bits.



Review

Operands of the Computer Hardware

- LEGv8 Register size – 64 Bits
- Total of **32 registers** (64-bit)
- Register name convention use **X** as prefix.
- Registers are names
  - X0
  - X1
  - …
  - X30
  - XZR(X31) (Exception, more on this later…!)



Review

Example – 3 (Again)

$$f = (g + h) - (i + j)$$

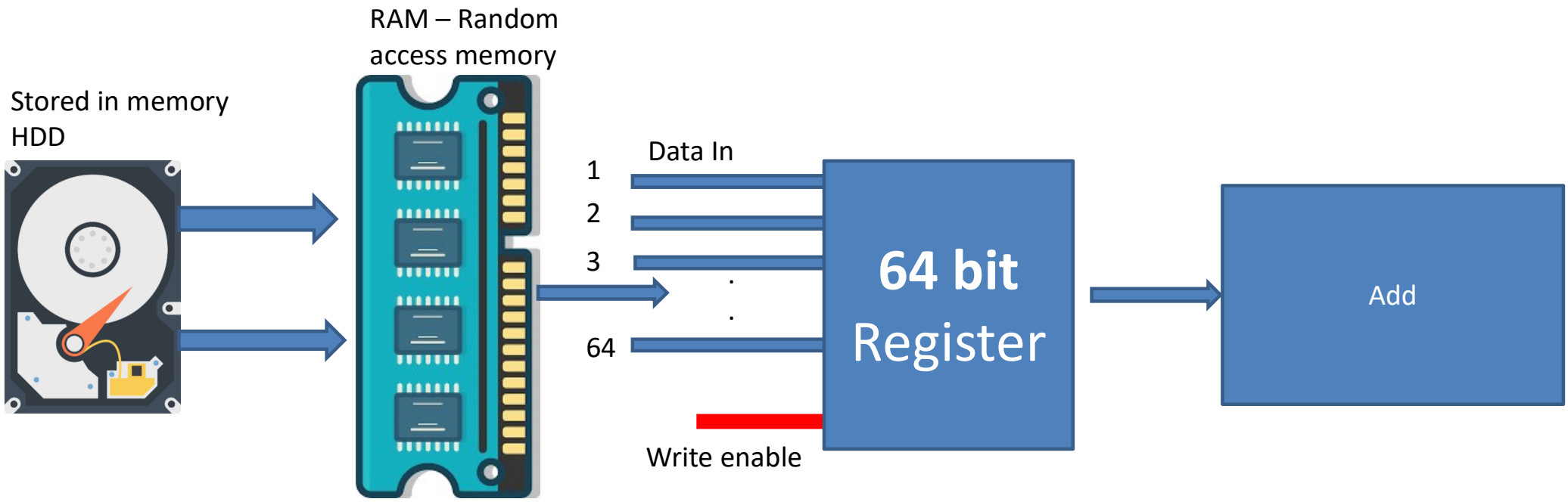
$$f, \ldots, j \; store \; in \; registers \; X19, X20, \ldots, X23$$

$$Two \; temporary \; registers \; are \; availabe \; X9 \; \& \; X10$$

```
ADD X9, X20, X21   // X9 = g + h
ADD X10, X22, X23 // X10 = i + j
SUB X19, X9, X10  // f = X9 − X10
```

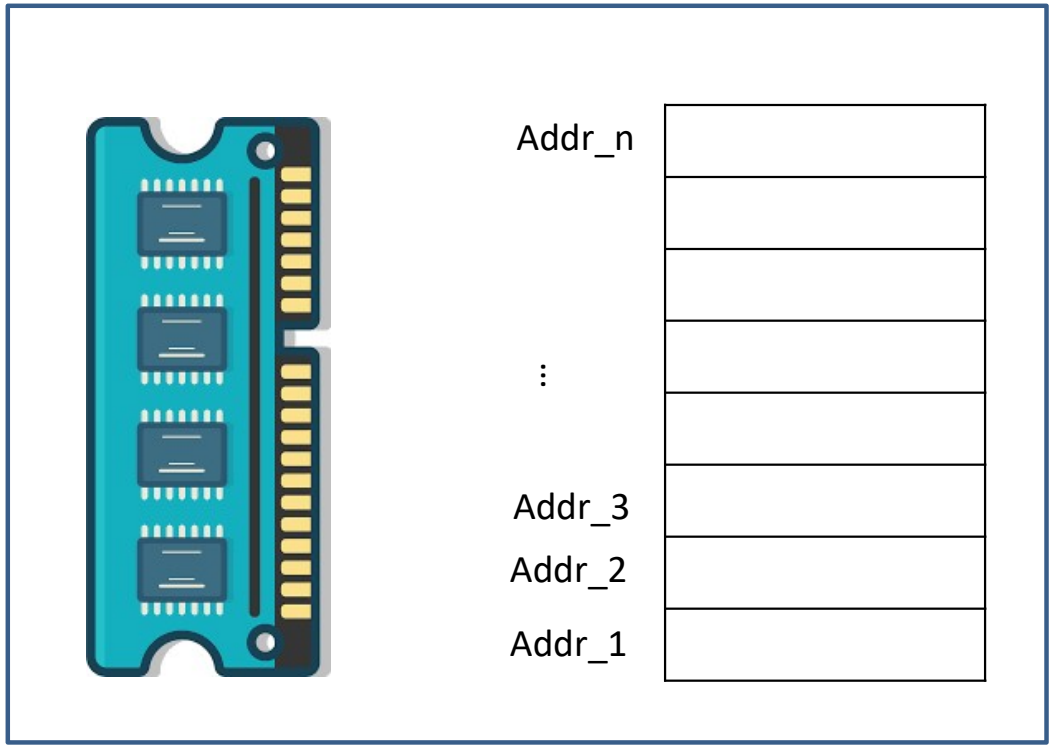
Review

Review: Half-Adder with manual input



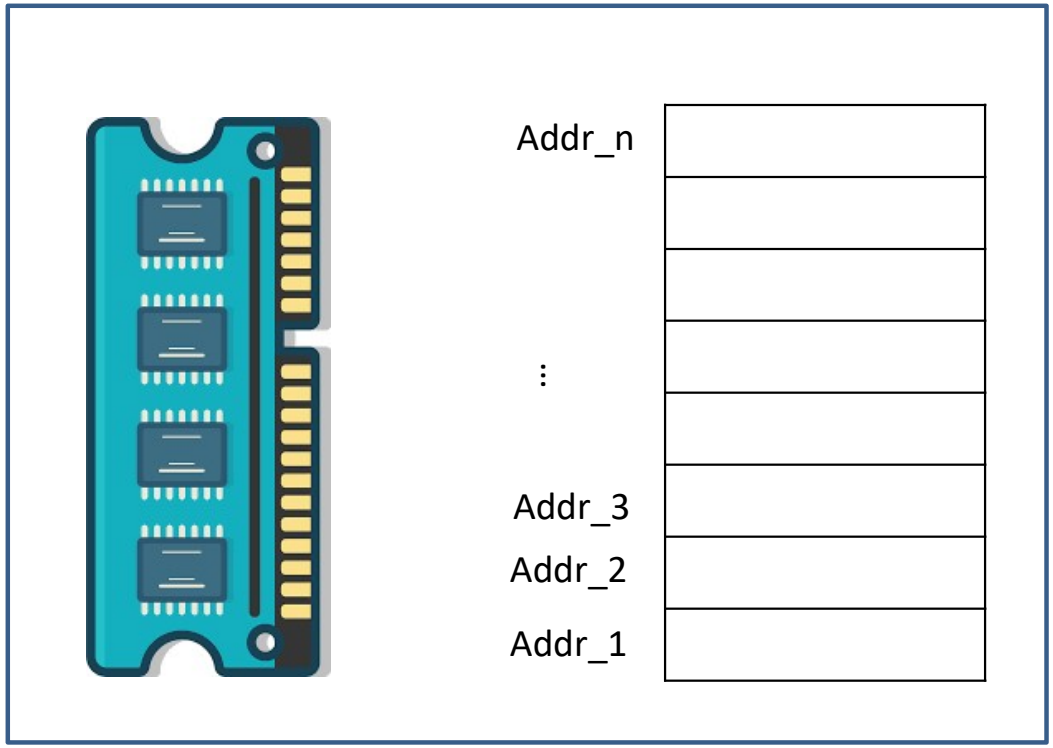
Review

Load Operation



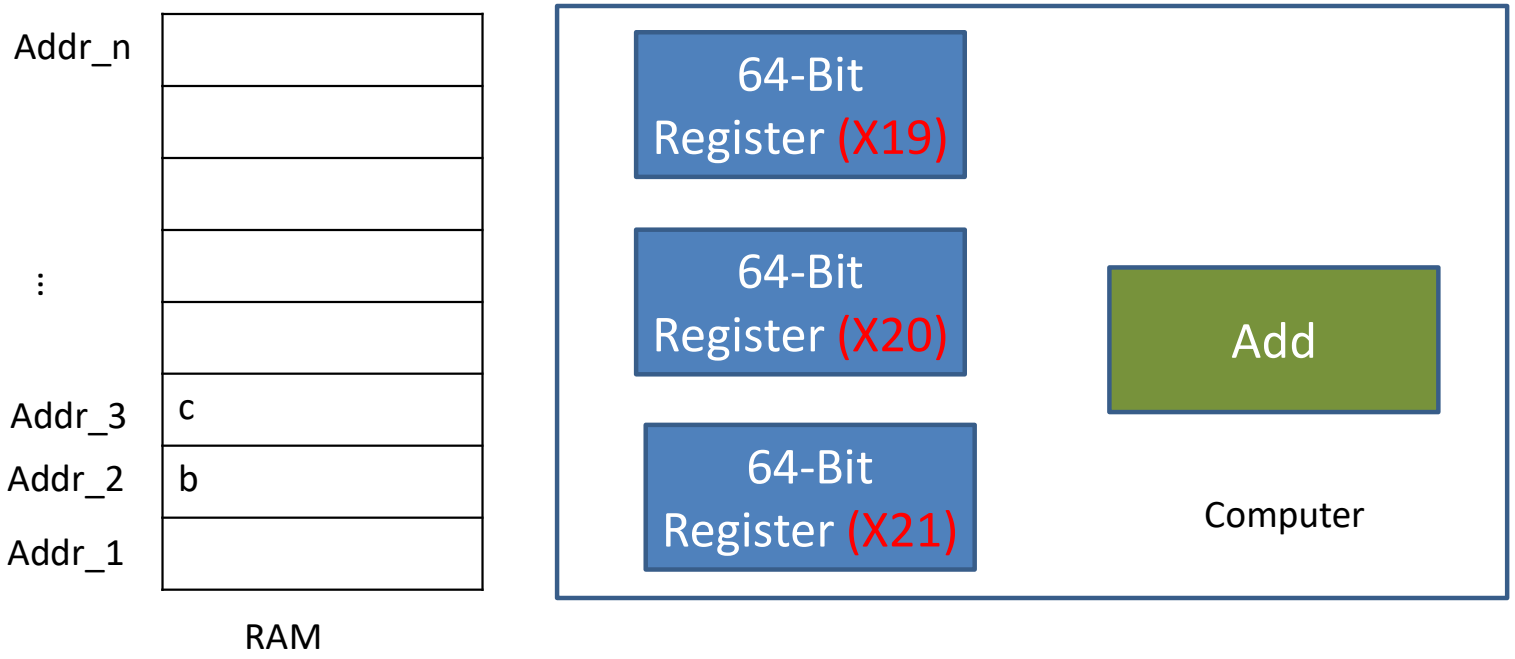
Review

Store Operation



Review

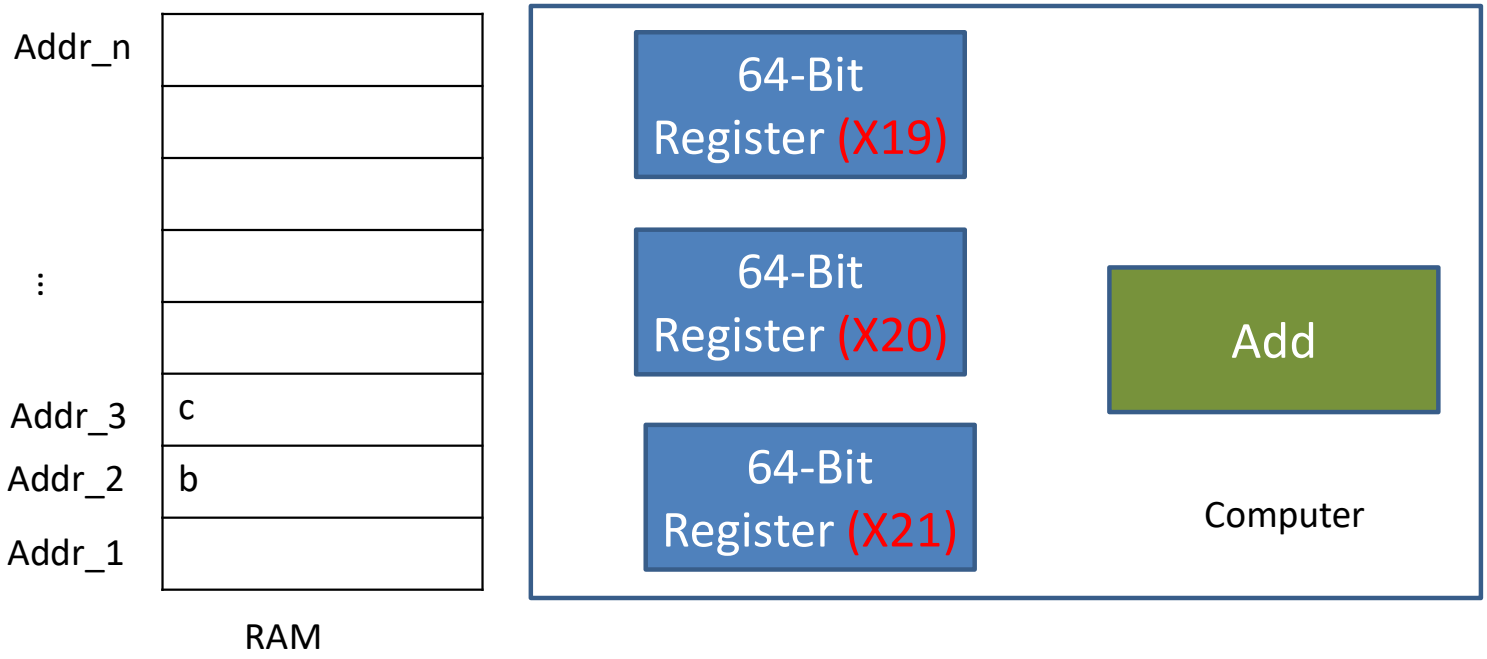
Memory Operand, LOAD



$a = b + c$

Review

Memory Operand , LOAD



$$a = b + c$$

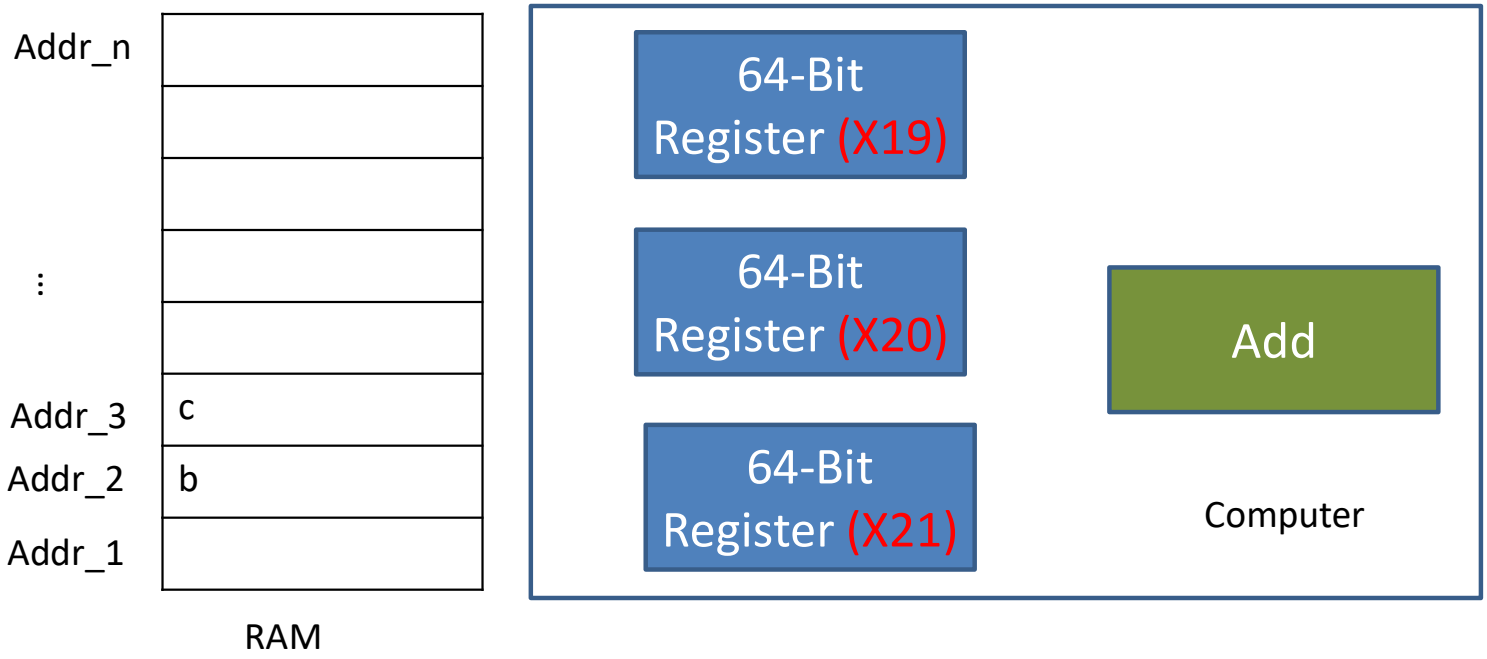
Load Addr_3 (c) to register X19
Load Addr_2 (b) to register X20
ADD X21, X19, X20

For Load:
Need to specify the ram memory address, and the register to load the value into.

memory address-> also in bits, and needs to be stored in another register.

Review

Memory Operand , LOAD



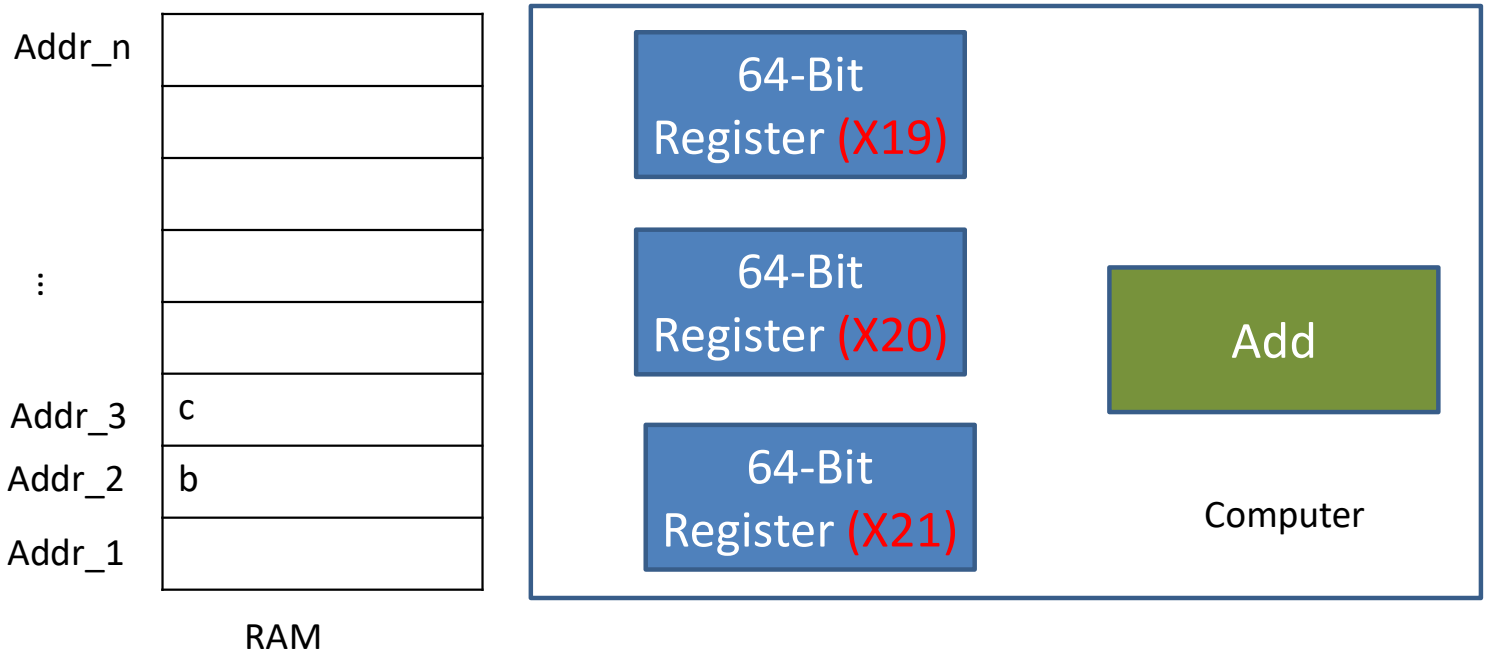
$$a = b + c$$

Load Addr_3 (c) to register X19
Load Addr_2 (b) to register X20
ADD X21, X19, X20

Lets assume memory address is stored in X22

Review

Memory Operand , LOAD



$$a = b + c$$

Load Addr_3 (c) to register X19
Load Addr_2 (b) to register X20
ADD X21, X19, X20

Lets assume memory address is stored in X22

The **LEGv8 instruction** to:
Load Addr_2 (a) to register X19

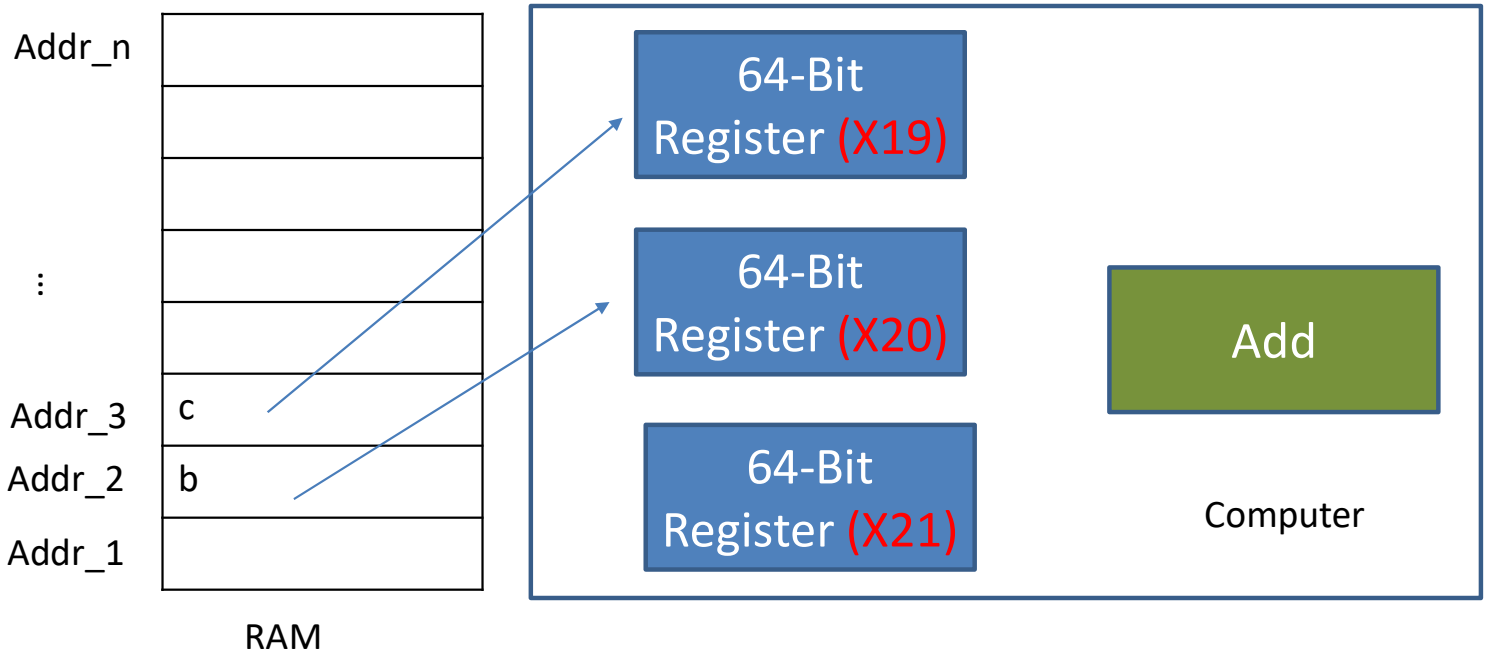
$$LDUR\ X19, [X22, \#const]$$

Destination register

Ram address

Review

Memory Operand , LOAD



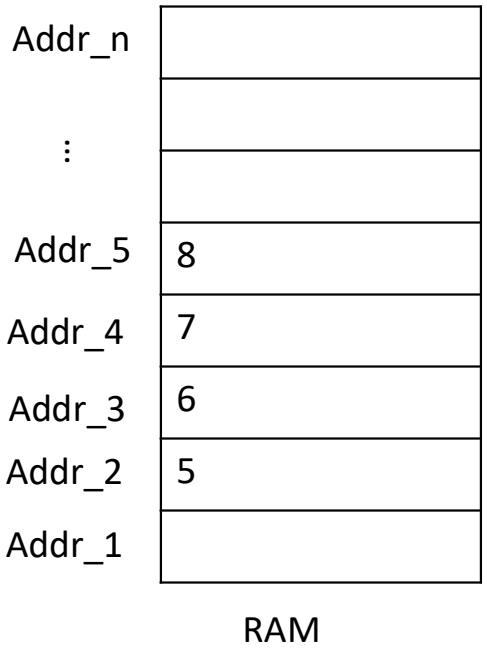
$a = b + c$

Assuming Addr_3 is store in X22 and Addr_2 is store in X23
Assembly code **(LEGv8 instruction)**

$LDUR\ X19, [X22, \#\mathbf{0}]$
$LDUR\ X20, [X23, \#\mathbf{0}]$
$ADD\ X21, X19, X20$

Review

Arrays in RAM



$int\ a[4] = \{5, 6, 7, 8\};$

1. Arrays are stored in contiguous memory

Let a start from Addr_2

a[0] ➔Addr_2

a[1] ➔ Addr_3

a[2] ➔Addr_4

a[3] ➔Addr_5

To load $a[1]$, we would have to specify where a starts in the memory, the offset (which is 1) and the destination register (d_register) to load it to.

Instruction

Load d_register, [addr_2, offset(1)]

A constant is needed to specify the offset to load arrays in the LDUR instruction

Review

Bits, Bytes, Words, and Double Words

For our course!!!

0 ➔ 1 **bit** of data
1 ➔ 1 **bit** of data

10011101 (8 bits) ➔ 1 **byte** of data

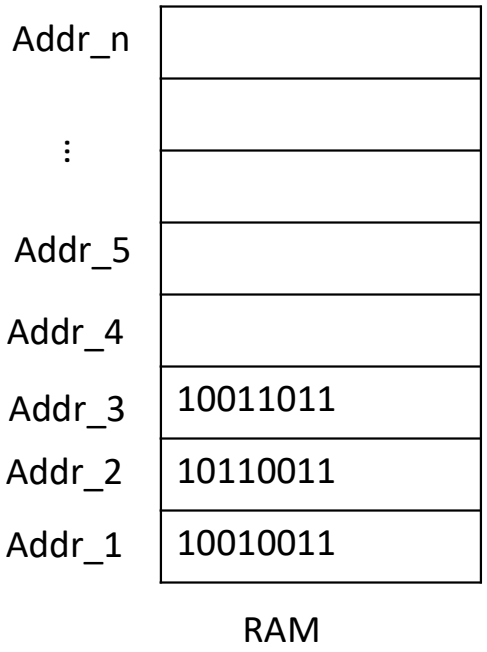
Overtime this became a **basic unit of data.**
Older system represented letters using bytes
As a results most memory hardware

10011101 **1 byte** 10010001 **2 byte** 10010101 **3 byte** 10010101 **4 byte** ➔ **4 bytes** is a **word** (32 bits)

10011101 **1 byte** 10010001 **2 byte** 10010101 **3 byte** … 10010101 **7 byte** ➔ **8 bytes** is a **Doubleword** (64 bits)

Review

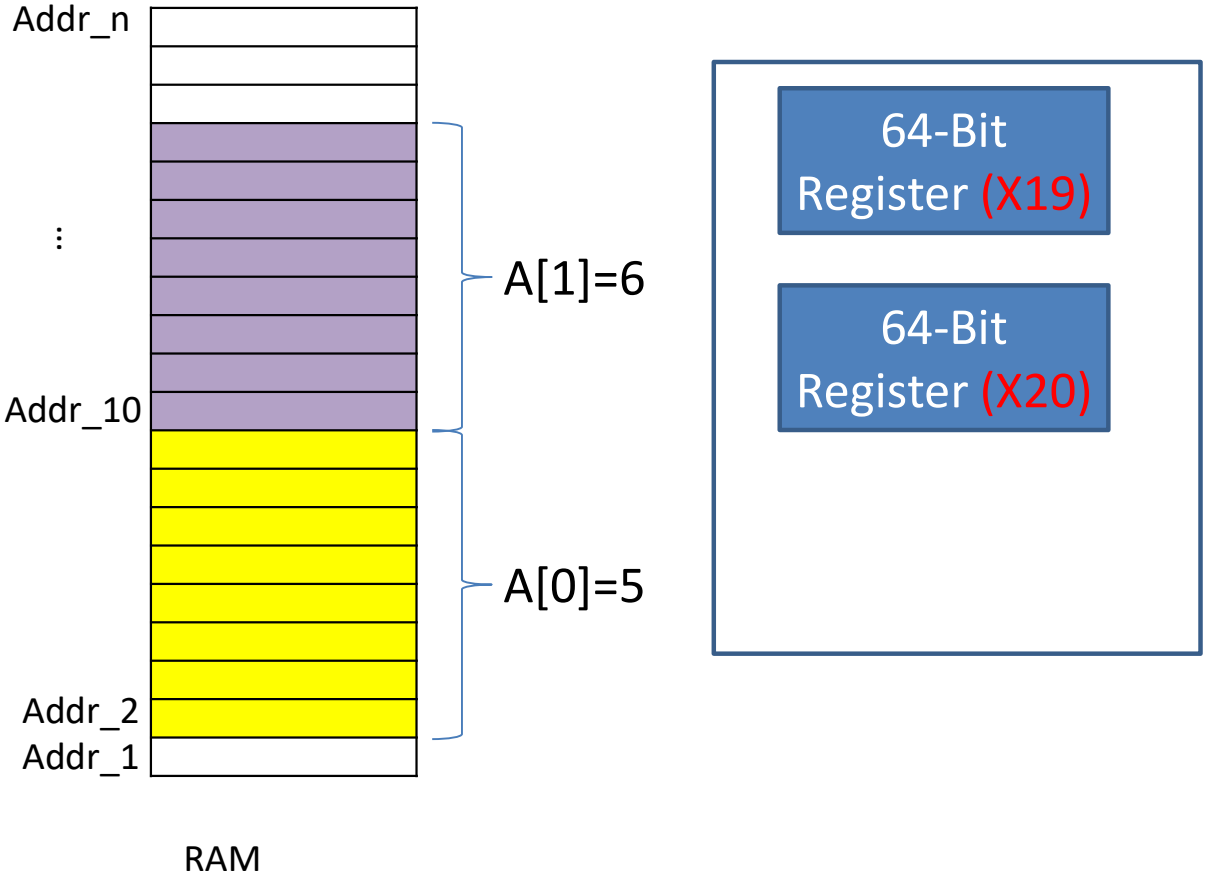
RAMS and Byte Addresses



1. Byte is considered a basic unit of data.
2. Most memory hardware store 1 byte of data at each address.
3. Each address is referred to as a **byte address**, as 8 bits are stores.

Review

Memory Operand , LOAD



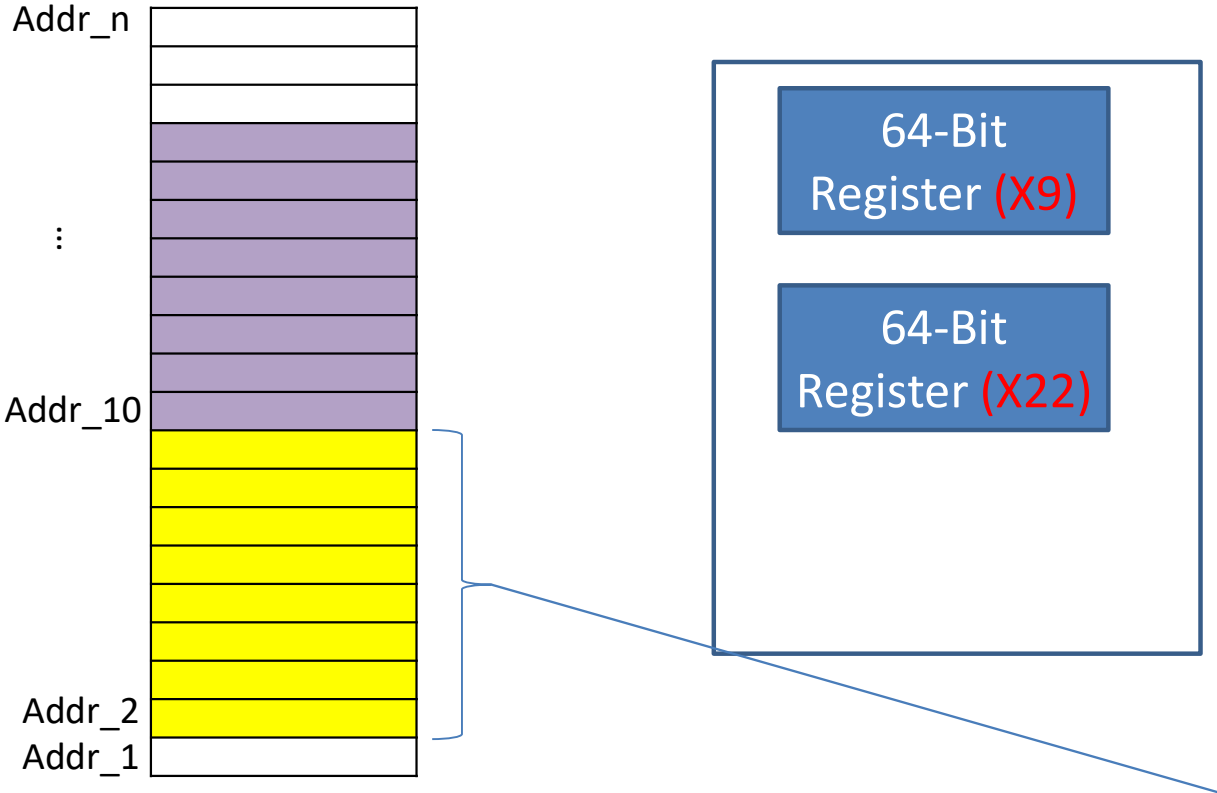
$int\ a[4] = \{5, 6, 7, 8\};$

Arrays are store in contiguous memory.
So
5 is stored using 64 bits
6 is stored using 64 bits

Let start address be Addr_2
Let Addr_2 be stored in register X22
What is the LEGv8 instruction to load a[0] into register X19?

Review

Memory Operand , LOAD



$int \quad [4] = \{5, 6, 7, 8\};$

Let start address be Addr_2
Let Addr_2 be stored in register X22
What is the LEGv8 instruction to load a[0] into register X9?

$LDUR\ X9, [X22, \#\mathbf{0}]$

What is the LEGv8 instruction to load a[1] into register X9?

$LDUR\ X9, [X22, \#8]$

What is the LEGv8 instruction to load a[3] into register X9?

$LDUR\ X9, [X22, \#\mathbf{24}]$

3 X 8 = 24

Review

Memory Operand Example

- C code:

  $A[12] = h + A[8];$

  – h in X21, base address of A (i.e. A[0]) in X22

- LEGv8 code:

```
LDUR    X9,[X22,#64]
ADD     X9,X21,X9
STUR    X9,[X22,#96]
```

Constant or Immediate Operands

- Using a constant in operation.
- More than half of arithmetic instructions have constant (SPEC CPU2006).

x = x + 4

Let X be stored in register X22
If X20 is some base register, and the number 4 is stored in the memory at location AddrConst4

Constant or Immediate Operands

- Using a constant in operation.
- More than half of arithmetic instructions have constant (SPEC CPU).

x = x +4

Let X be stored in register X22
If X20 is some base register, and the number 4 is stored in the memory at location AddrConst4

Load 4 into register
Add

Constant or Immediate Operands

- Using a constant in operation.
- More than half of arithmetic instructions have constant (SPEC CPU2006).

Very common to use constants.
Too much time to load constants from memory.
Why not make a version of Add with one operand fixed to a specific value.
One operand is always 4.

x = x +4

Let X be stored in register X22
If X20 is some base register, and the number 4 is stored in the memory at location AddrConst4 (offset from X20)

LEGv8 Instructions:
$LDUR\ X9, [X20, \#AddrConst4]$
$ADD\ X22, X22, X9$

Constant or Immediate Operands

- Using a constant in operation.
- More than half of arithmetic instructions have constant (SPEC CPU2006).

Very common to use constants.
Too much time to load constants from memory.
Why not make a version of Add with one operand fixed to a specific value.
One operand is always 4.

x = x +4

Let X be stored in register X22
If X20 is some base register, and the number 4 is stored in the memory at location AddrConst4 (offset from X20)

LEGv8 Instructions:
$LDUR\ X9, [X20, \#AddrConst4]$
$ADD\ X22, X22, X9$

LEGv8 Instructions:
$\boldsymbol{ADDI}\ X22, X22, \#4$
Add immediate