Computer Organization and Architecture

COSC 2425

Lecture – 4

Aug 31st, 2022

Acknowledgement: Slides from Edgar Gabriel & Kevin Long

Review

Energy Consumed by Transistor

Dominant technology is CMOS (complementary metal oxide semiconductor.
The primary source for energy consumption is called dynamic energy

0 → 1 → 0 1 → 0 → 1

$$Energy \propto \text{Capacitive load} \times \text{Voltage}^2$$

Energy consumed by transistor to switch states.

0 → 1 1 → 0

$$Energy \propto 0.5 \times \text{Capacitive load} \times \text{Voltage}^2$$

Review

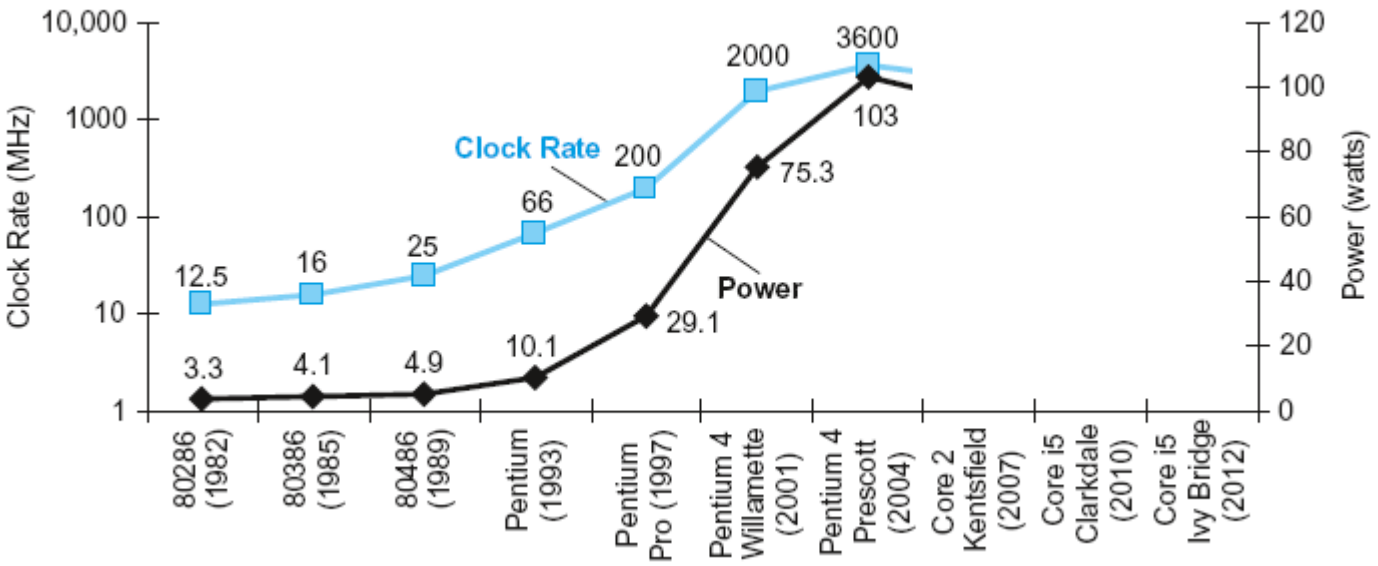
Power required per transistor

- Depends on number of transitions, frequency of the cpu

$$Power \propto 0.5 \times \text{Capacitive load} \times \text{Voltage}^2 \times Frequency\ Switched$$

- Capacitive load depends on
  - Number of transistors connected to output
  - Technology (defines capacitance of wires, and transistors)

Power Trends



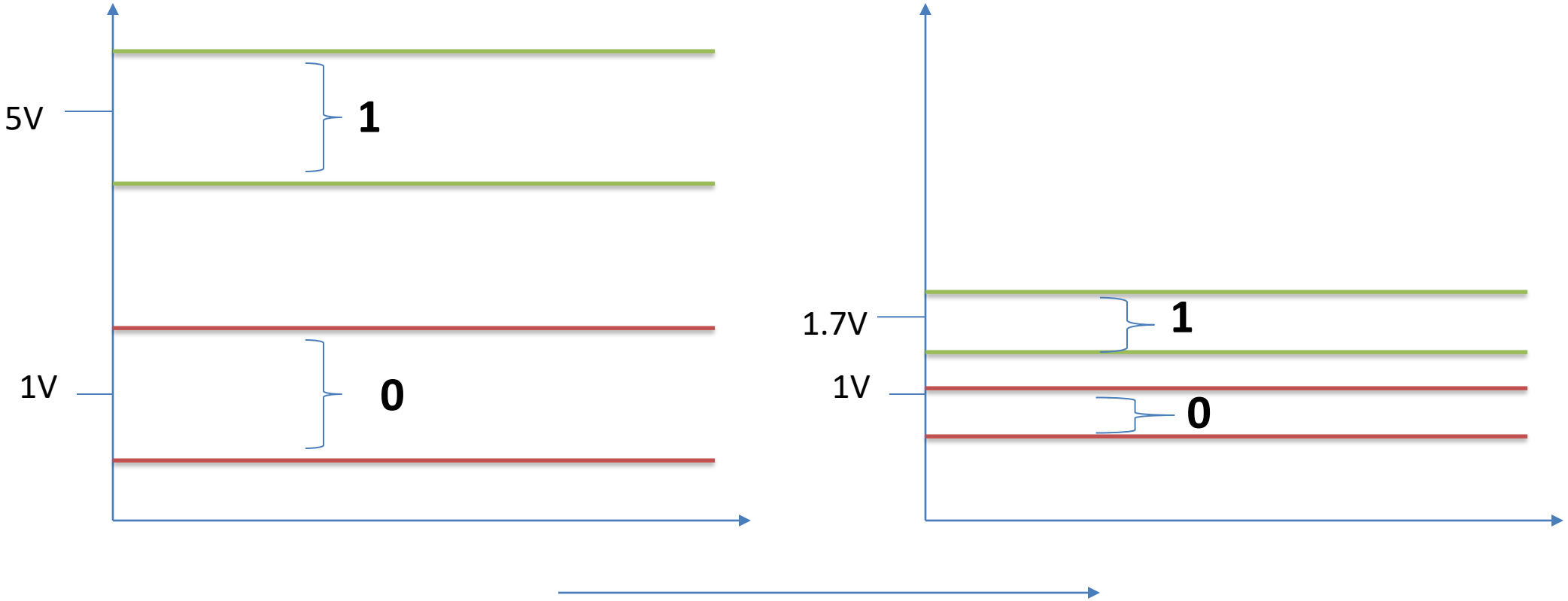
$$Power \propto 0.5 \times \text{Capacitive load} \times \text{Voltage}^2 \times Frequency\ Switched$$

×30

×1000

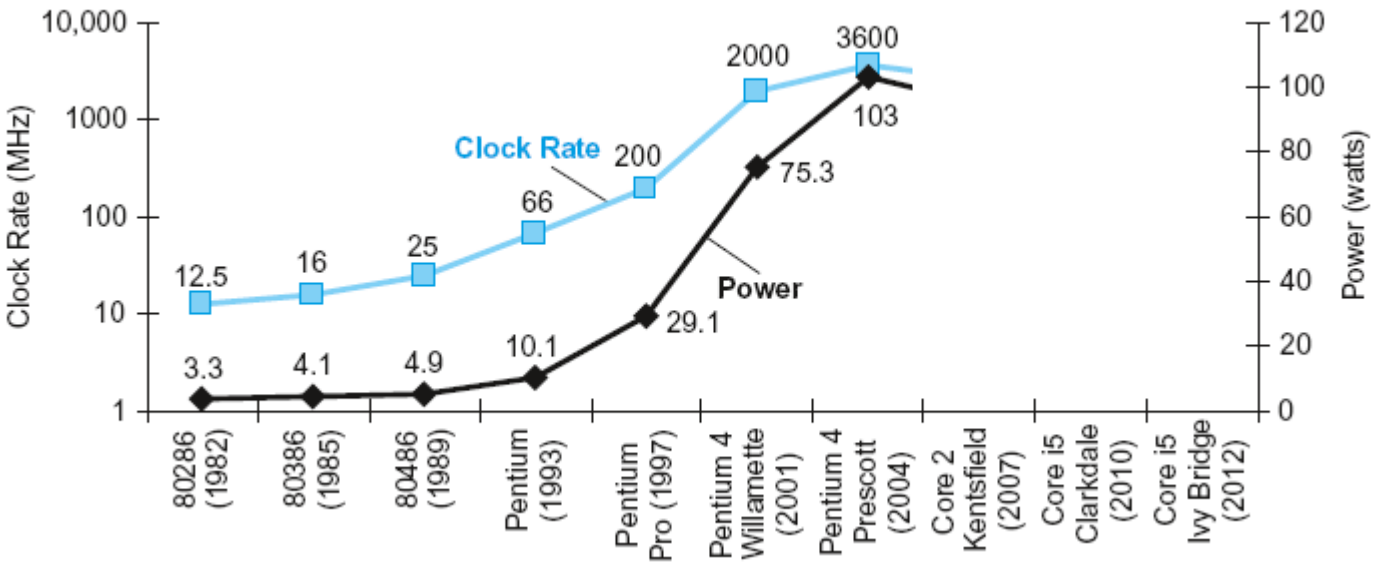
Reducing Voltage

Reduces overall power consumptions



Improvements in technology allowed for lesser fluctuations and lowering the voltage

Power Trends



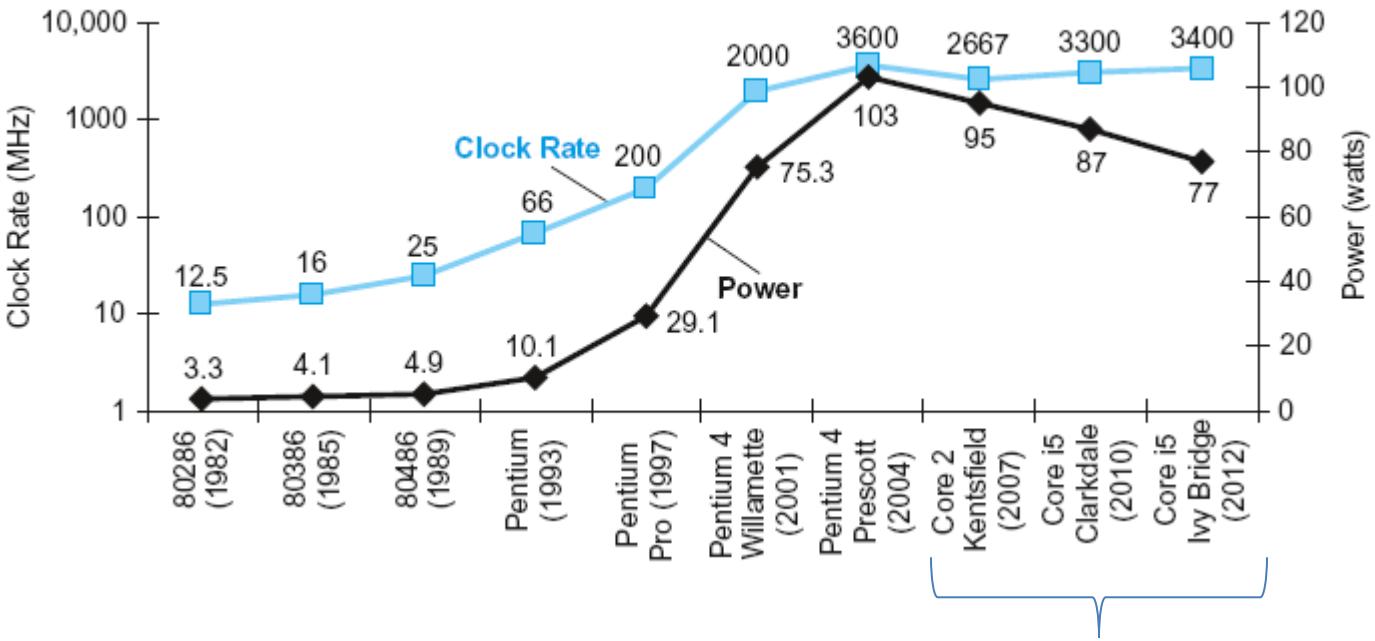
$$Power \propto 0.5 \times \text{Capacitive load} \times \text{Voltage}^2 \times Frequency\ Switched$$

×30

5V → 1V

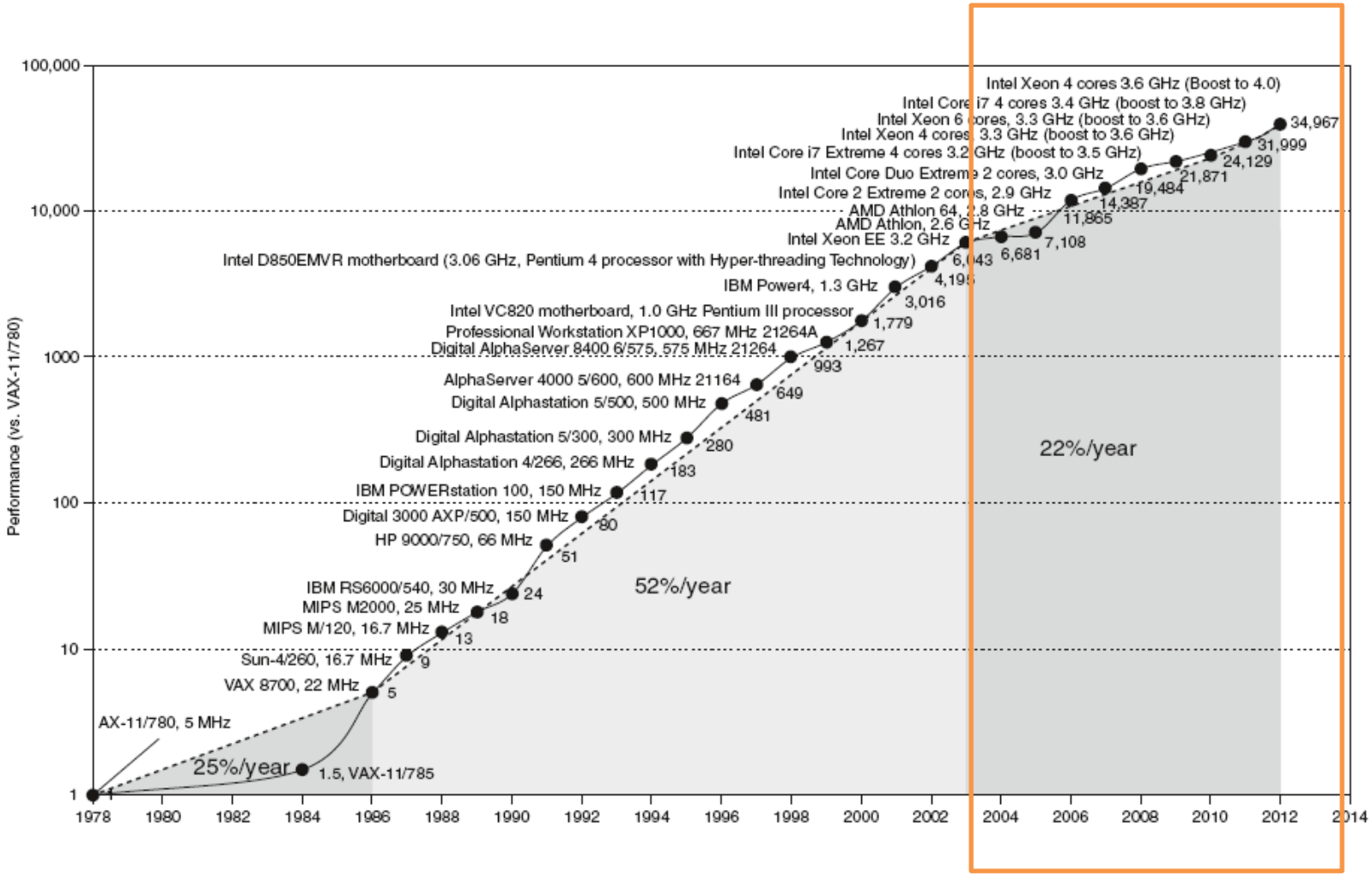
×1000

Power Wall



Multiple processors per chip

Response time



- Instead of reducing response time, focused on increasing throughput.
- 2006 and later computers shipped with multiple processes (per chip)

Reduced from 1.5 per year to 1.2 per year

Multiprocessors

- Multicore microprocessors
  - More than one processor per chip
- Requires explicitly parallel programming
  - Compare with instruction level parallelism
    - Hardware executes multiple instructions at once
    - Hidden from the programmer
  - Hard to do
    - Programming for performance
    - Load balancing
    - Optimizing communication and synchronization
      - Scheduling, load balancing, synchronization, communication (overhead)

SPEC CPU Benchmark (Updated)

- Programs used to measure performance
  - Supposedly typical of actual workload
- **Standard** Performance Evaluation Corp (SPEC)
  - Develops benchmarks for CPU, I/O, Web, …
- SPEC **CPU2017**
  - Elapsed time to execute a selection of programs
    - Negligible I/O, so focuses on CPU performance

CINT2006 for Intel Core i7 920

Description	Name	Instruction Count x 10^9	CPI	Clock cycle time (seconds x 10^{-9})	Execution Time (seconds)	Reference Time (seconds)	SPECratio
Interpreted string processing	perl	2252	0.60	0.376	508	9770	19.2
Block-sorting compression	bzip2	2390	0.70	0.376	629	9650	15.4
GNU C compiler	gcc	794	1.20	0.376	358	8050	22.5
Combinatorial optimization	mcf	221	2.66	0.376	221	9120	41.2
Go game (AI)	go	1274	1.10	0.376	527	10490	19.9
Search gene sequence	hmmer	2616	0.60	0.376	590	9330	15.8
Chess game (AI)	sjeng	1948	0.80	0.376	586	12100	20.7
Quantum computer simulation	libquantum	659	0.44	0.376	109	20720	190.0
Video compression	h264avc	3793	0.50	0.376	713	22130	31.0
Discrete event simulation library	omnetpp	367	2.10	0.376	290	6250	21.5
Games/path finding	astar	1250	1.00	0.376	470	7020	14.9
XML parsing	xalancbmk	1045	0.70	0.376	275	6900	25.1
Geometric mean	–	–	–	–	–	–	25.7

Performance Summary

$$\sqrt[n]{\prod_{i=1}^{n} \text{Execution time ratio}_i}$$

Pitfall: Amdahl's Law

$$T_{improved} = \frac{T_{affected}}{\text{improvement factor}} + T_{unaffected}$$

- Example:
- Takes a total of 100s
- multiply accounts for 80s (of the 100s)
  - How much improvement in multiply performance to get 5× overall?

$$(5\ times\ faster)100/5 = \ 20 = \frac{80}{n} + 20$$

- Can't be done!

Pitfall: MIPS as a Performance Metric

- MIPS: Millions of Instructions Per Second
  - Doesn't account for
    - Differences in ISAs between computers
    - Differences in complexity between instructions

$$\text{MIPS} = \frac{\text{Instruction count}}{\text{Execution time} \times 10^6}$$

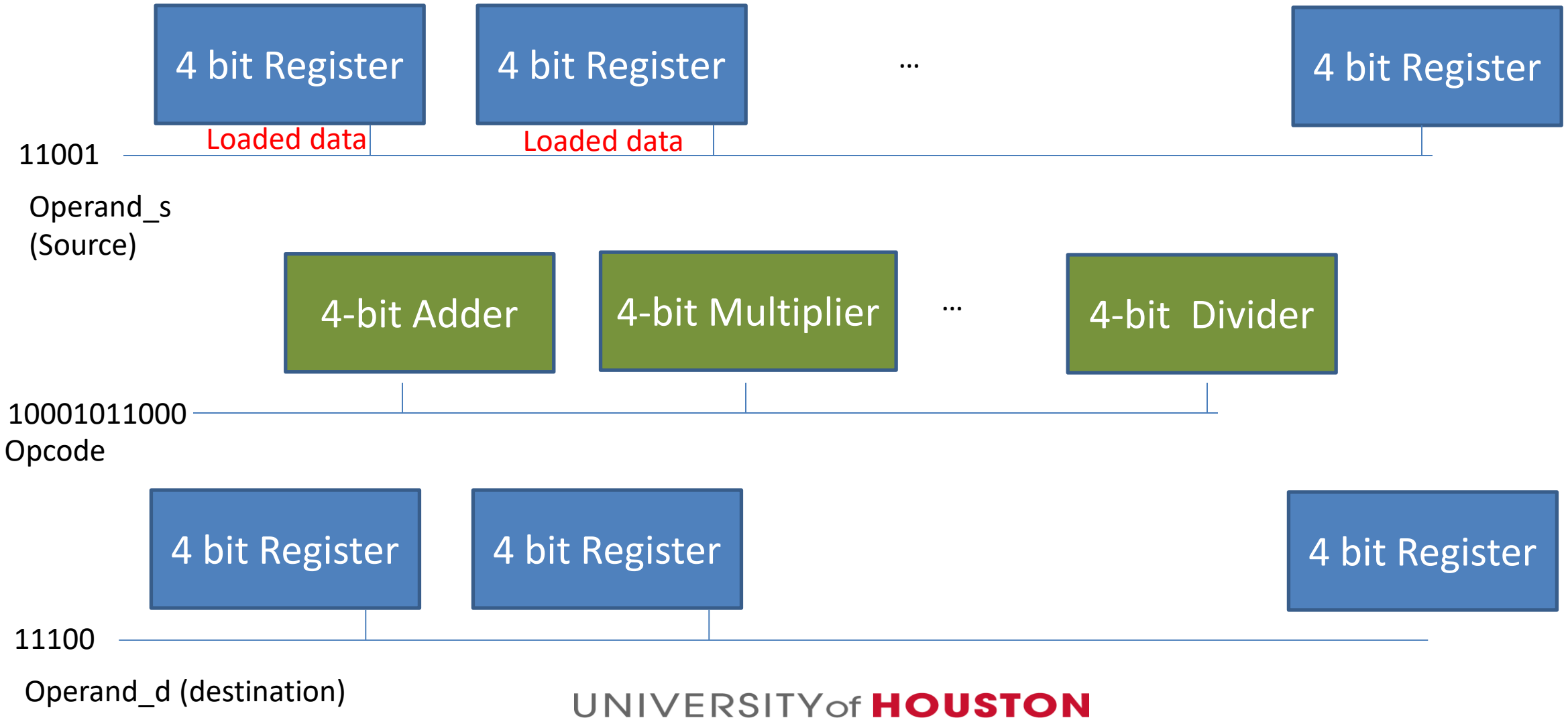
$$= \frac{\text{Instruction count}}{\dfrac{\text{Instruction count} \times \text{CPI}}{\text{Clock rate}} \times 10^6} = \frac{\text{Clock rate}}{\text{CPI} \times 10^6}$$

- CPI varies between programs on a given CPU

Chapter 2

Instructions: Language of the Computer

Instruction



Instruction Example

Opcode	Operand_s1	Operand_s2	Operand_d
10001011000	11001	11010	11100

Instruction : 10001011000 11001 11010 11100

Instruction are represented in binary form. Stored in memory.
The only language a computer understand.
Byte code, machine code, …

Instruction Set

- Add
- Multiply
- Divide
- Load Data

Instruction Set

Instruction Set

- Add
- Multiply
- Divide
- Load Data

Instruction Set

Computer 1

ISA1

Computer 2

ISA2

A manual to instruct he computer.

Many Popular Instruction Sets

- …
- …
- MIPS
- ARMv7
- ARMv8
- …

1. Intentionally designed with simplicity as a design principle. More on this…

2. Have similar instruction sets, the hardware has similar underlying technology.

3. Have a common goal, to find language to build hw and compilers to maximize performance.

Many Popular Instruction Sets

- …
- …
- MIPS
- ARMv7
- ARMv8
- …

Commonly used in embedded systems (Simple architecture), Bluetooth chips, routers, etc.

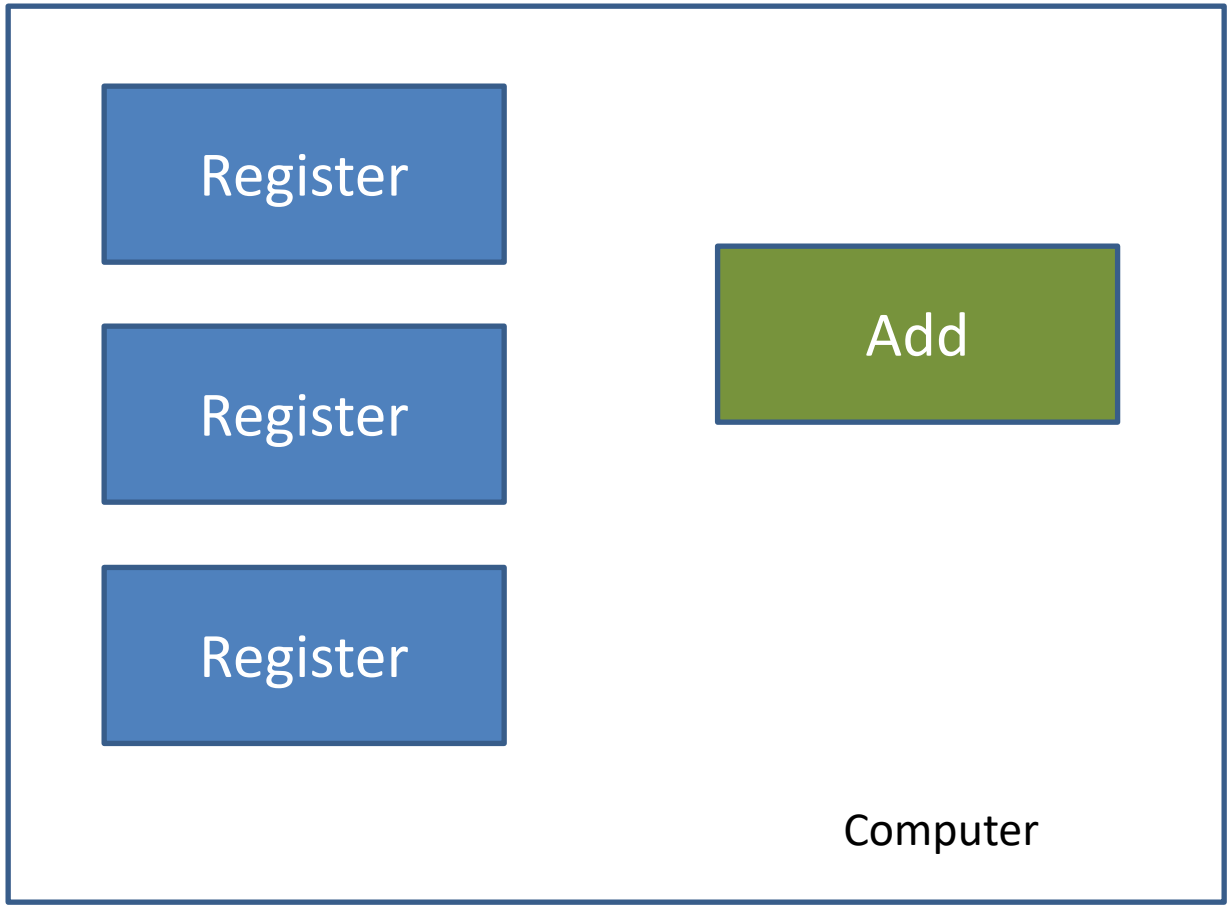
Commonly used computers. Desktops, Laptops, etc.

LEGv8: A subset of ARMv8 instructions, used for teaching purpose.

The ARMv8 Instruction Set

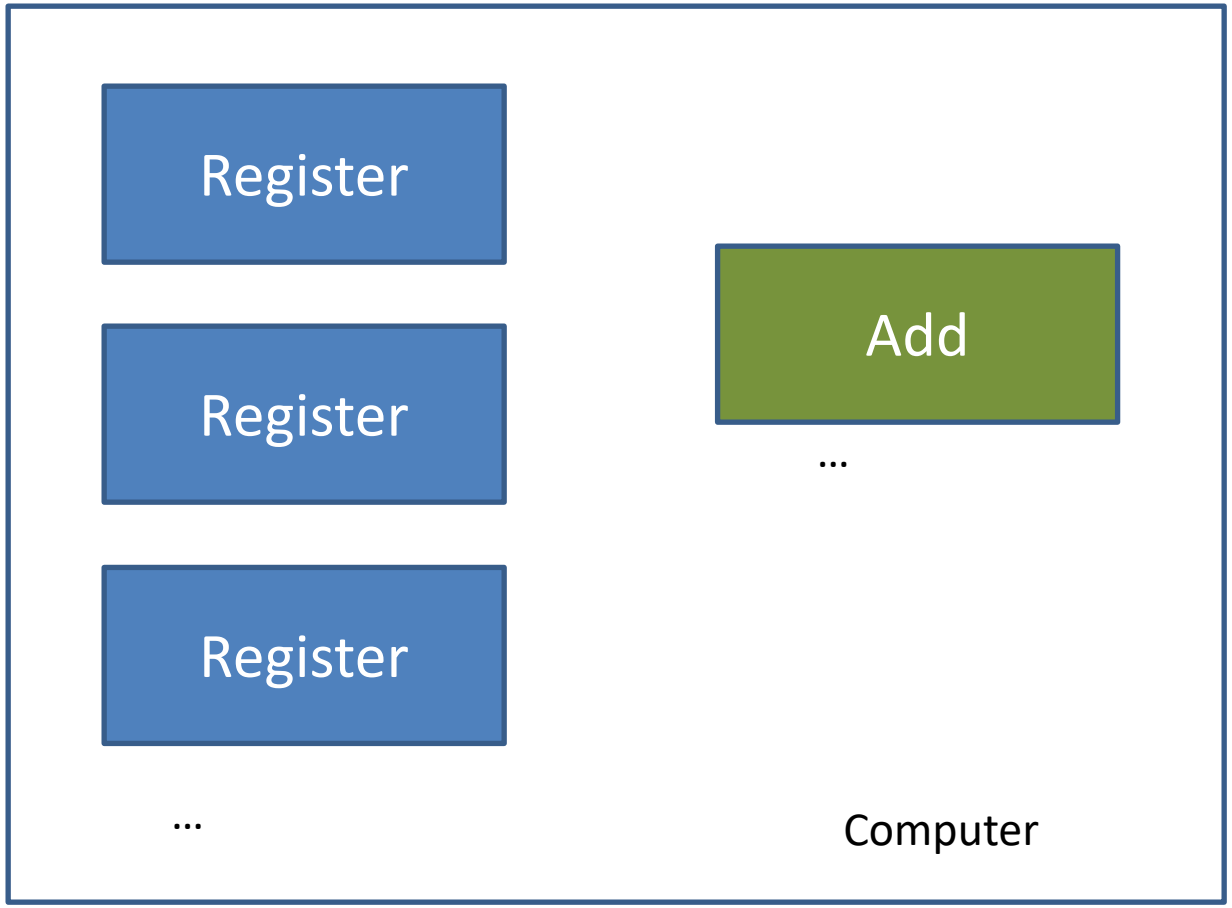
- A subset, called LEGv8, used as the example throughout the book
- Commercialized by ARM Holdings (www.arm.com)
- Large share of embedded core market
  - Applications in consumer electronics, network/storage equipment, cameras, printers, …
- Typical of many modern ISAs
  - See ARM Reference Data tear-out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

Operations of Computer Hardware



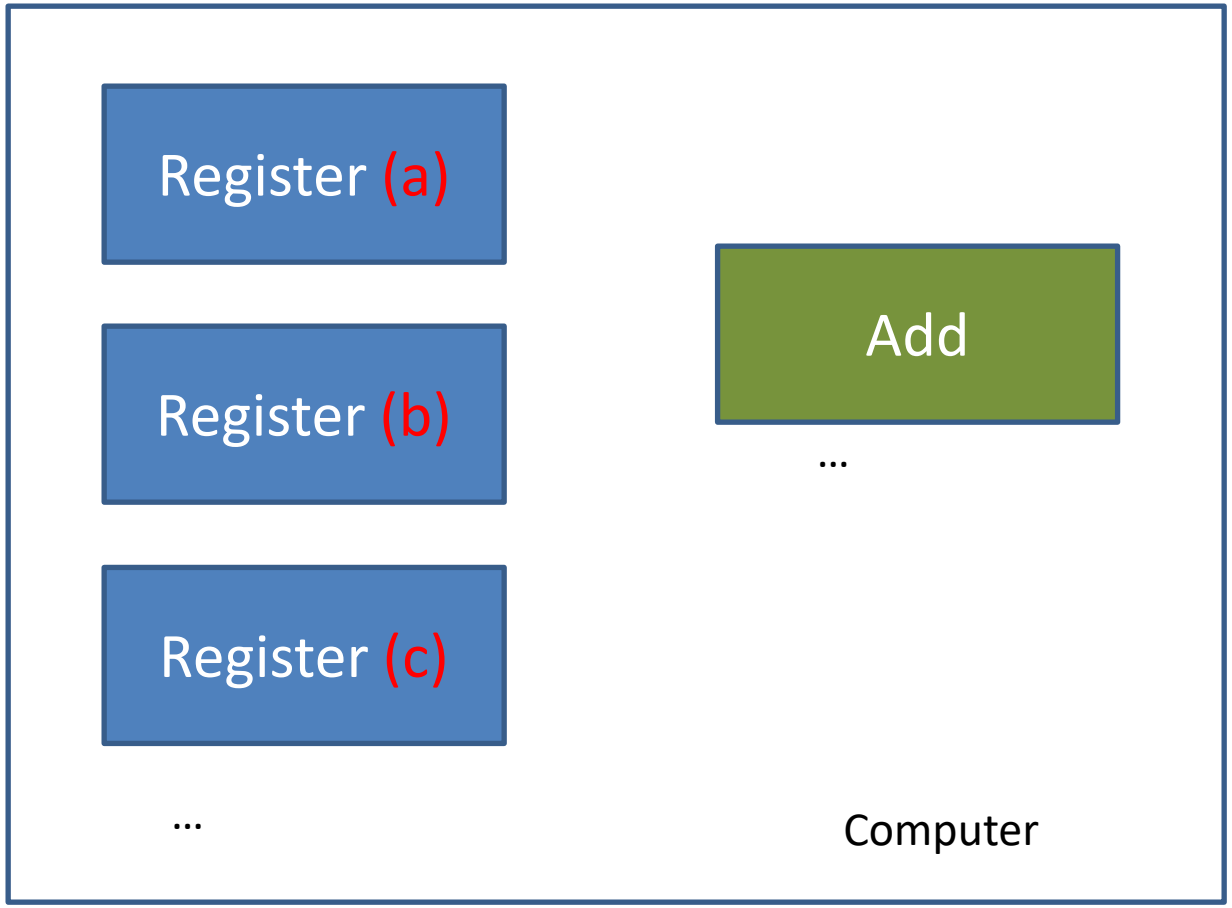
1. Has registers, and logic gate to perform operations. E.g. add
2. Add can only access data in the registers

Operations of Computer Hardware



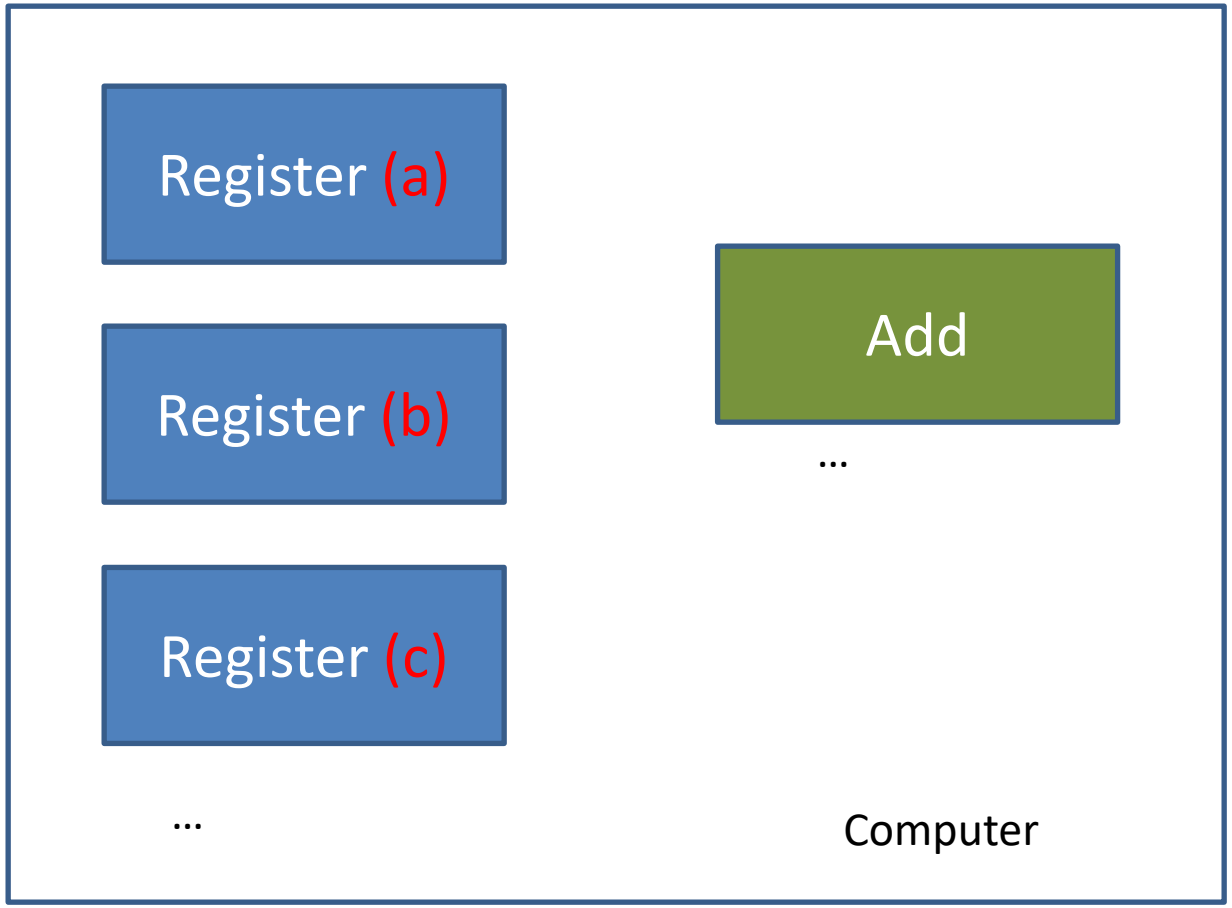
1. Has multiple registers, and logic gates to perform operations. E.g. add
2. Registers contain/store data.
3. Operators (like Add), can only access data in the registers.

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.

Operations of Computer Hardware



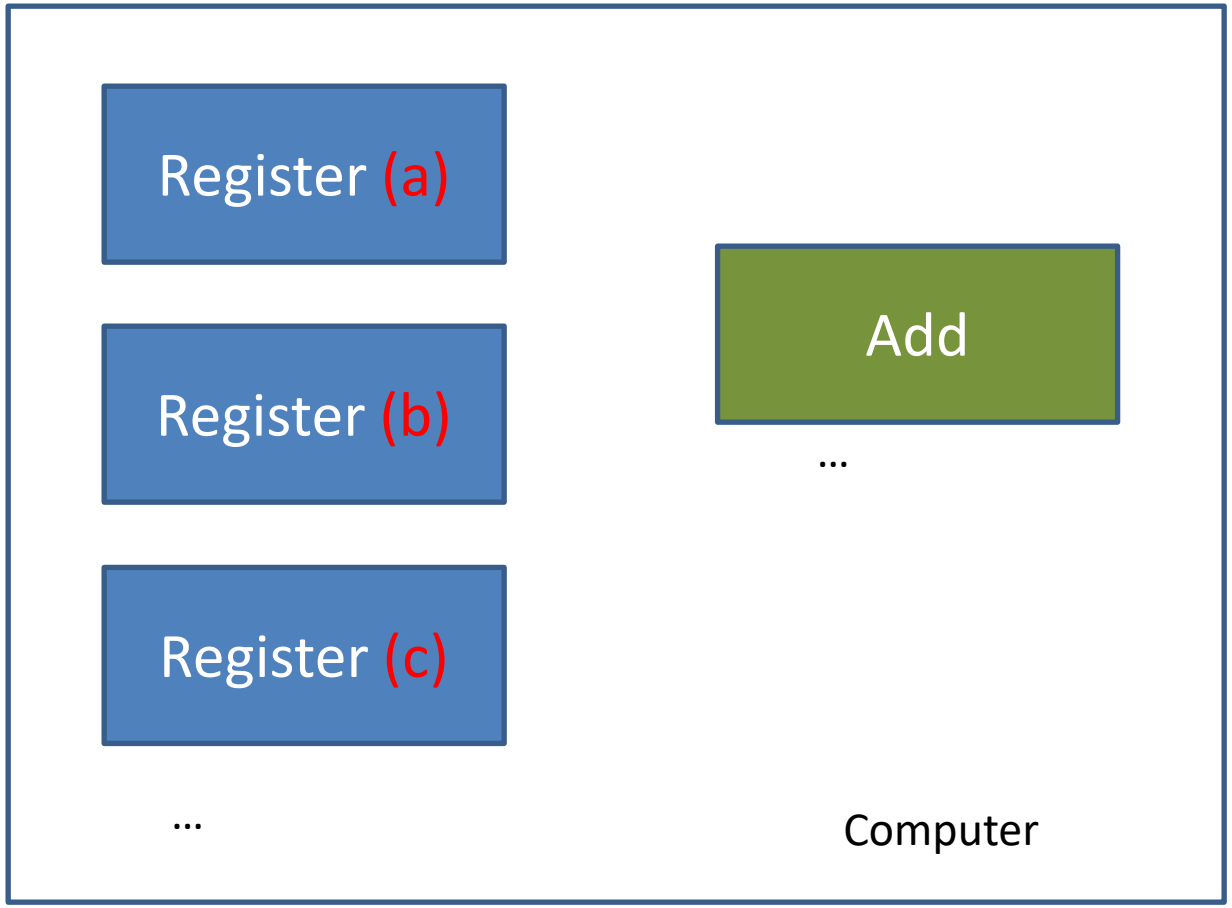
1. Has multiple registers, and logic gates to perform operations. E.g. add
2. Registers contain/store data.
3. Operators (like Add), can only access data in the registers.

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:

$$ADD\ a, b, c$$

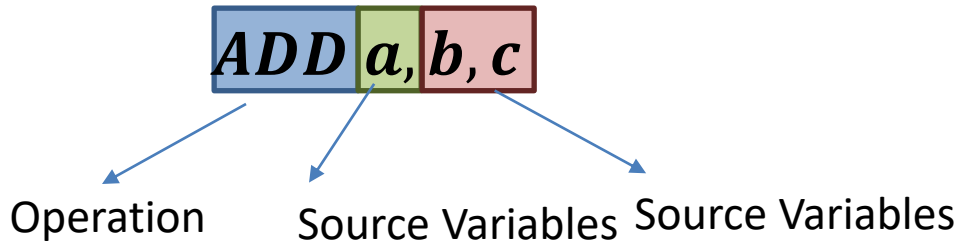
Operations of Computer Hardware



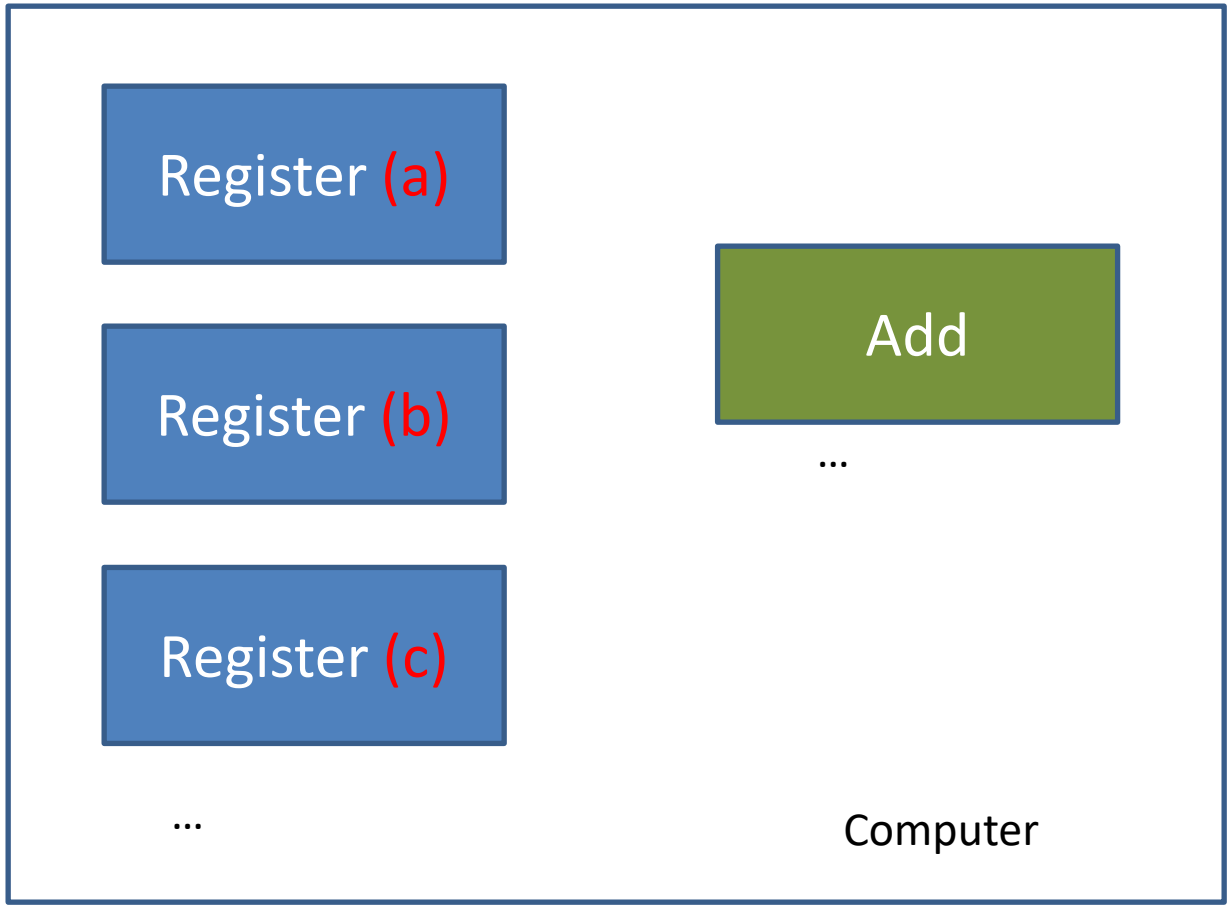
1. Has multiple registers, and logic gates to perform operations. E.g. add
2. Registers contain/store data.
3. Operators (like Add), can only access data in the registers.

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:



Operations of Computer Hardware

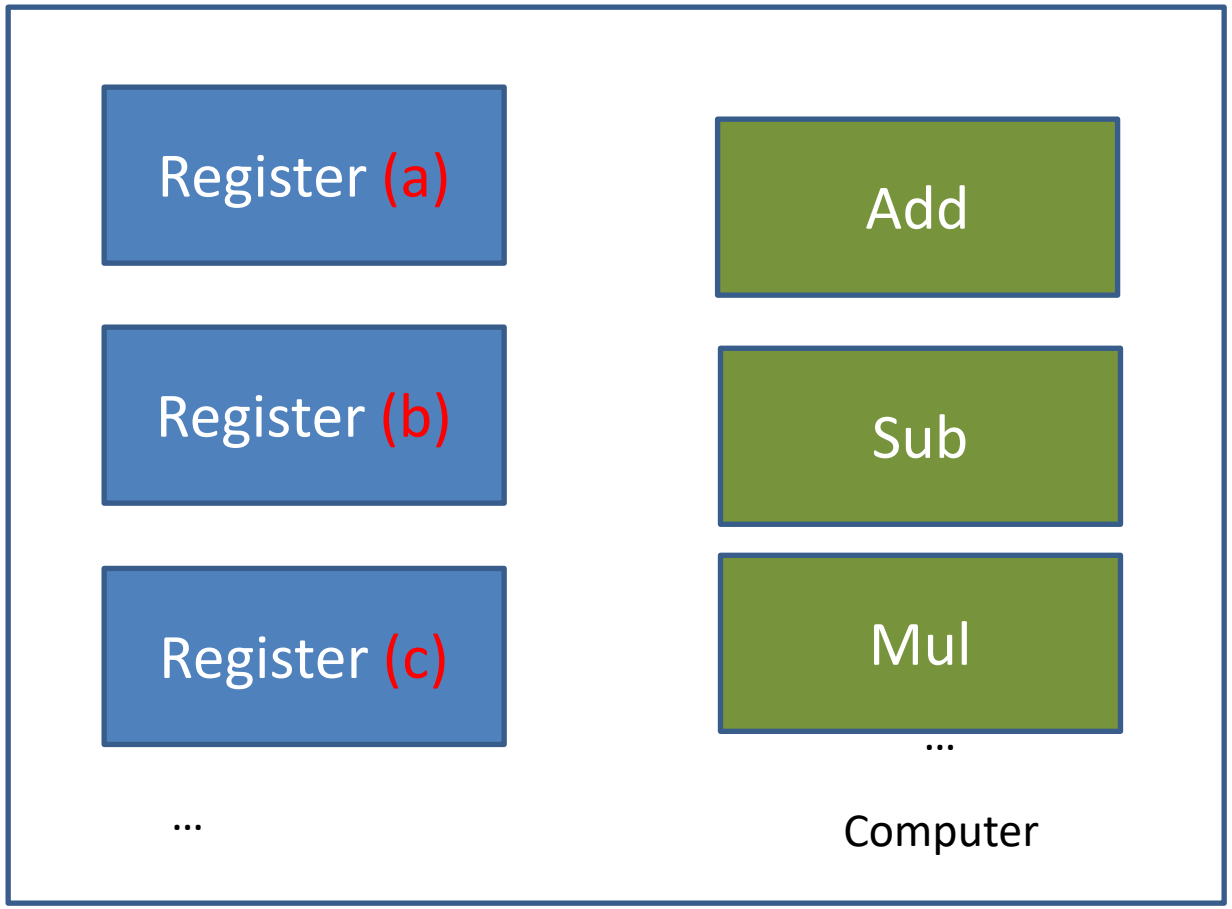


1. To instruct computer to
   1. **Add** (operation)
   2. **Values** in register **b and c** (Source Variables)
   3. Store the **result in a** (Destination Variable)

LEGv8 Instruction:

$$\underbrace{ADD}_{\text{One operation}}\ \underbrace{a, b, c}_{\text{Has three variables}}$$

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:

$$\underbrace{\boldsymbol{ADD}}_{\text{One operation}}\ \underbrace{\boldsymbol{a, b, c}}_{\text{Has three variables}}$$

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 $(a = b\ * c)$

- $SUB\ a, b, c$ // subtract instrcution $(a = b\ - c)$

Comment

Design Principle 1: Simplicity favors regularity

1. Regularity makes implementation simpler
2. Simplicity enables higher performance at lower cost

$$SUB\ \underbrace{a, b, c}$$

1. Three Variables, is natural number operands for arithmetic operations.
2. Requiring exactly three variables, keeps the hardware simple.
    1. Hardware for a variable number of operands is more complicated.

Example - 1

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

Example - 1

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

$$\boxed{\begin{array}{ll} ADD\ a, b, c & //\ a = b + c \\ ADD\ a, a, d & //\ a = a + d \\ ADD\ a, a, e & //\ a = a + e \end{array}}$$

3 instruction to sum 4 variables

Example - 2

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

Example - 2

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

$$\boxed{\begin{array}{ll} ADD\ a, b, c & //\ a = b + c \\ SUB\ d, a, e & //\ d = a - e \end{array}}$$

Example - 3

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

Example - 3

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

$$\boxed{\begin{array}{ll} ADD\ t0, g, h & //\ t0 = g + h \\ ADD\ t1, i, j & //\ t1 = i + j \\ SUB\ f, t0, t1 & //\ f = t0 - t1 \end{array}}$$

t0, t1: temporary variables created by the compiler

Example - 3

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

$$\boxed{\begin{array}{ll} ADD\ t0, g, h & //\ t0 = g + h \\ ADD\ t1, i, j & //\ t1 = i + j \\ SUB\ f, t0, t1 & //\ f = t0 - t1 \end{array}}$$

Variables $\underbrace{\text{t0, t1, f, g, h, i, j}}_{\textbf{Stored in registers}}$

Bits, Bytes, and Words

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

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

10011101	10010001	10010101	…	10010101 ➔ n bytes is a **word**
byte	**byte**	**byte**		**byte** (8*n bits)

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

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

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

Bits, Bytes, Words, and Double Words

For our course!!!

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

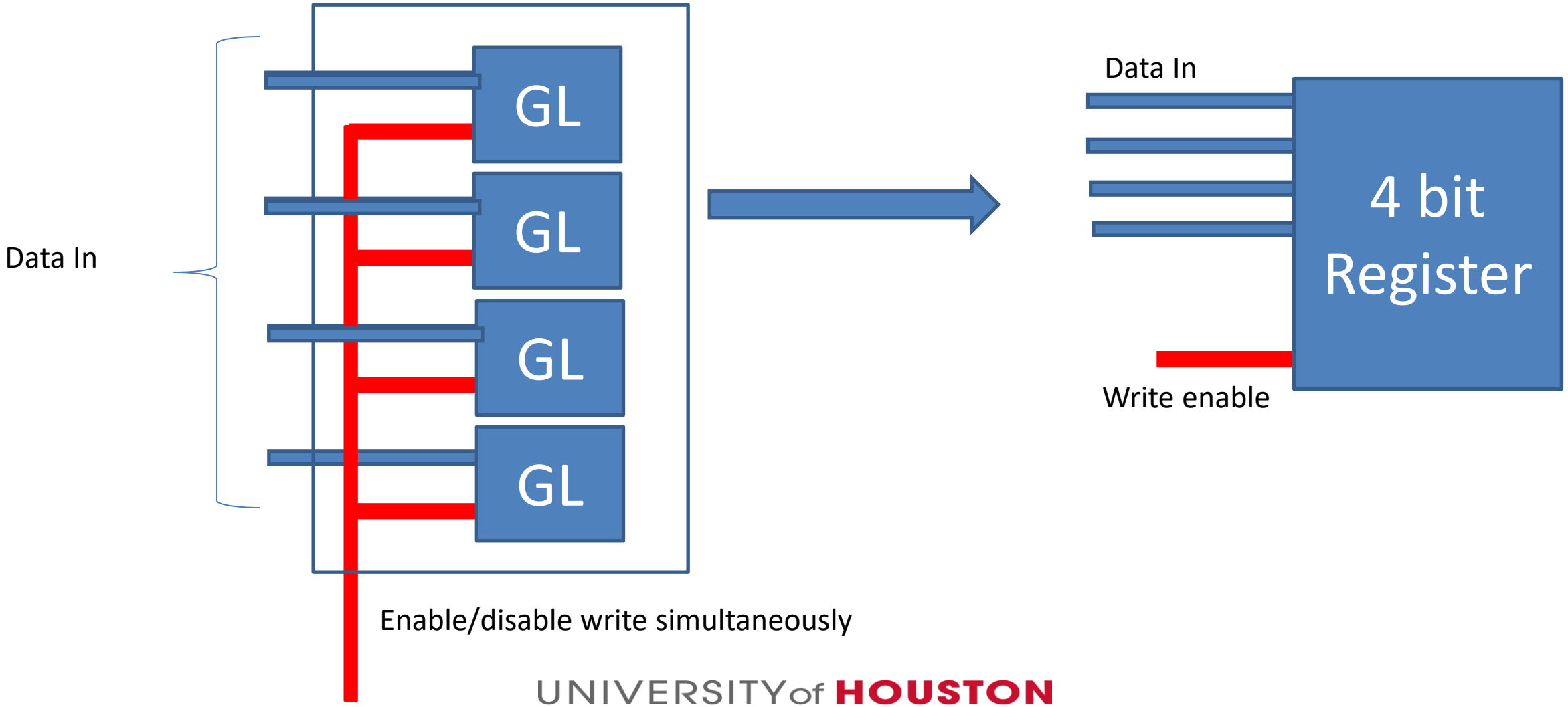
10011101 (8 bits) ➔ 1 **byte** of data

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

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

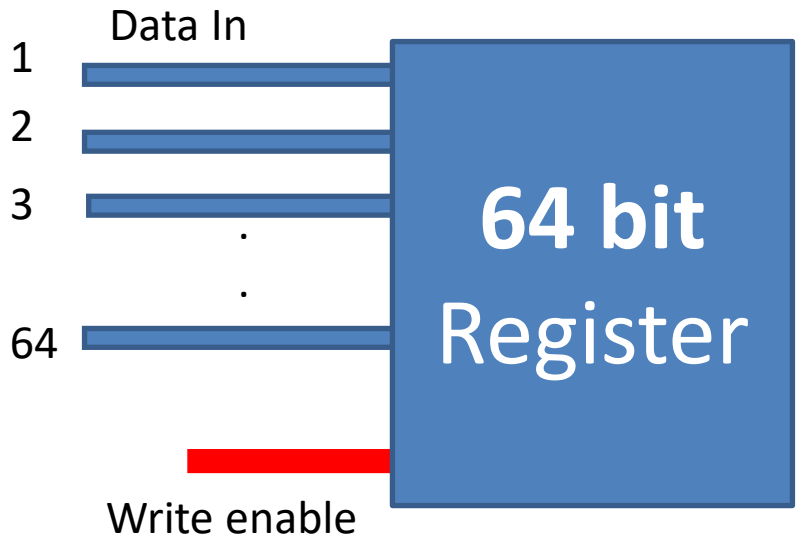
10011101	10010001	10010101	…	10010101 ➔ **8 bytes** is a **Doubleword**
1 byte	**2 byte**	**3 byte**		**7 byte** (64 bits)

Register (4 bit) – A group of latches



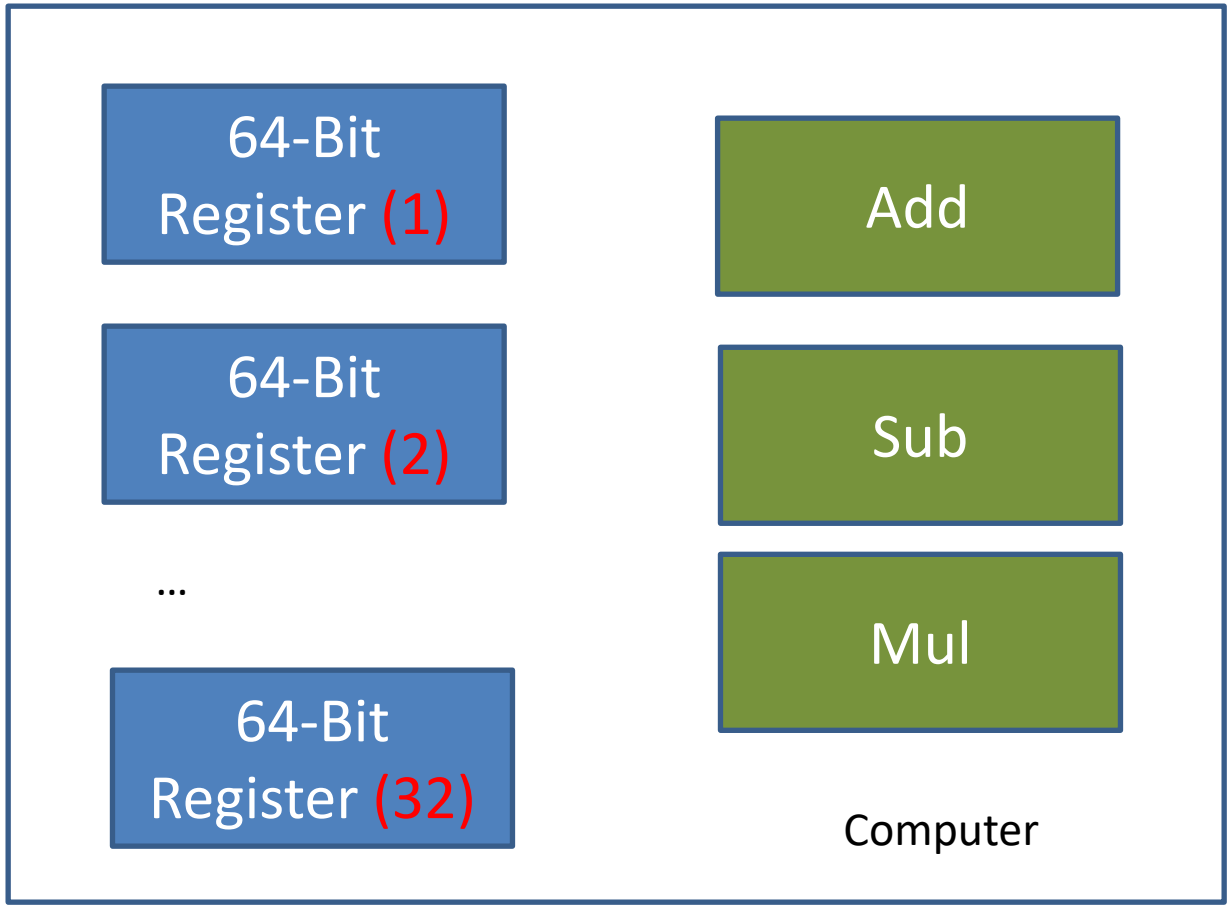
Operands of the Computer Hardware

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



Operands of the Computer Hardware

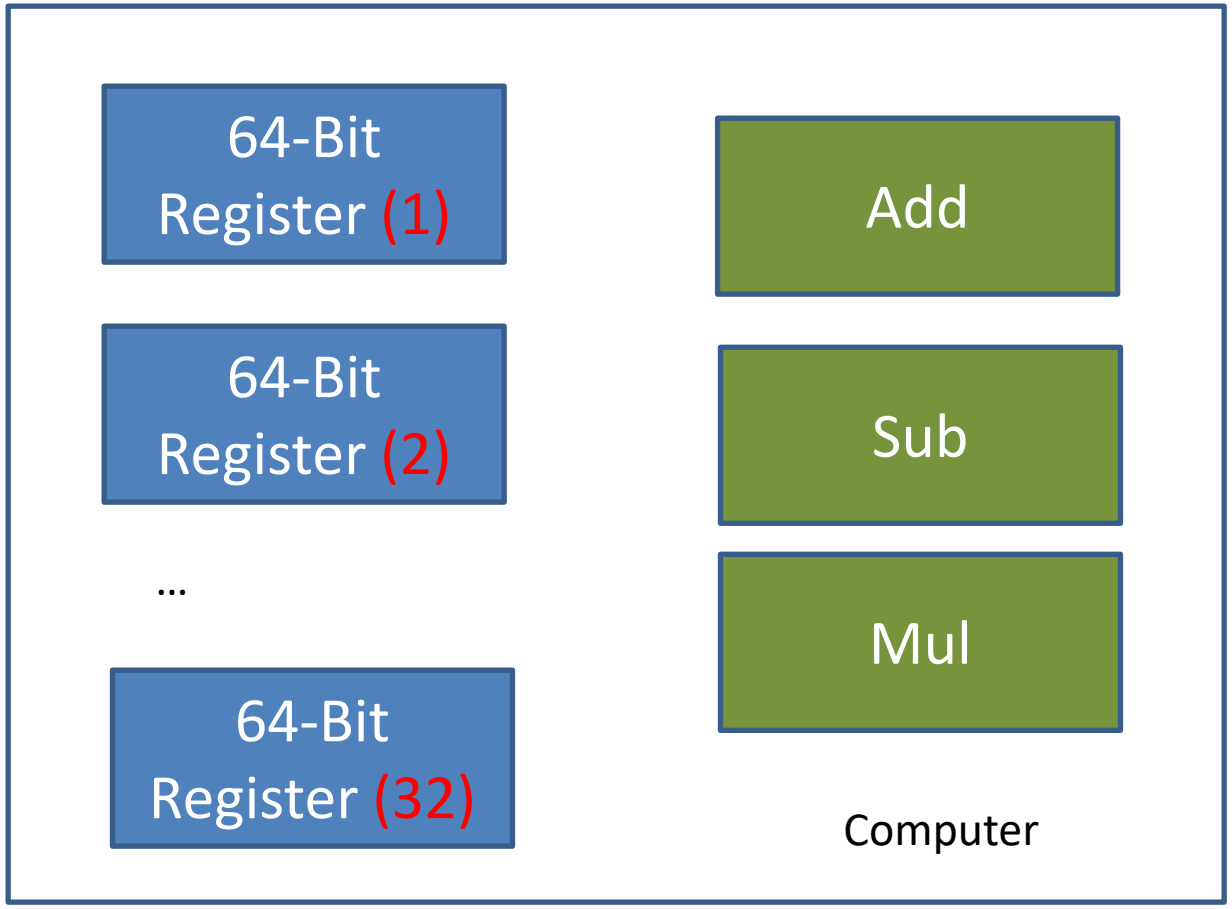
- LEGv8 Register size – 64 Bits
- Total of **32 registers** (64-bit)



Operands of the Computer Hardware

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

Why only 32??



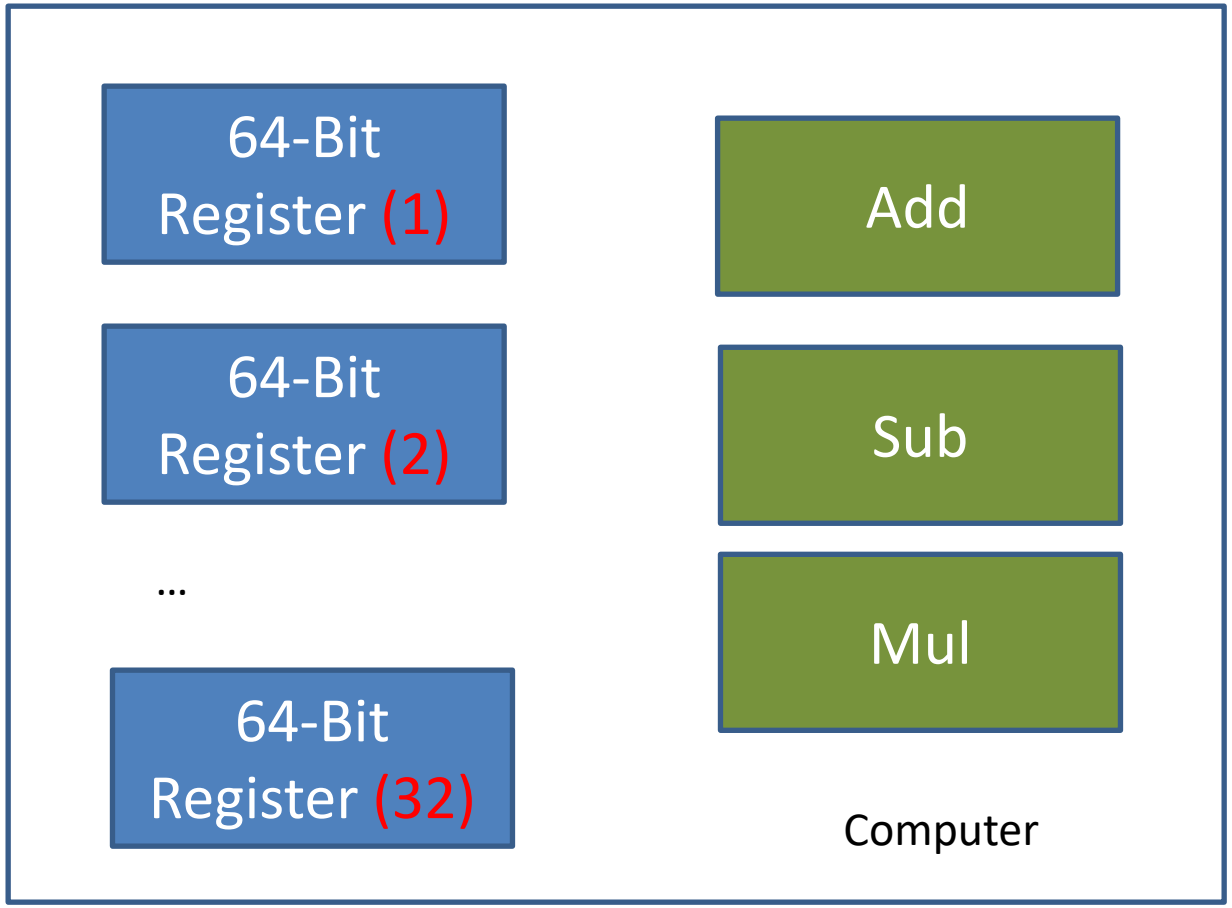
Operands of the Computer Hardware

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

Why only 32??

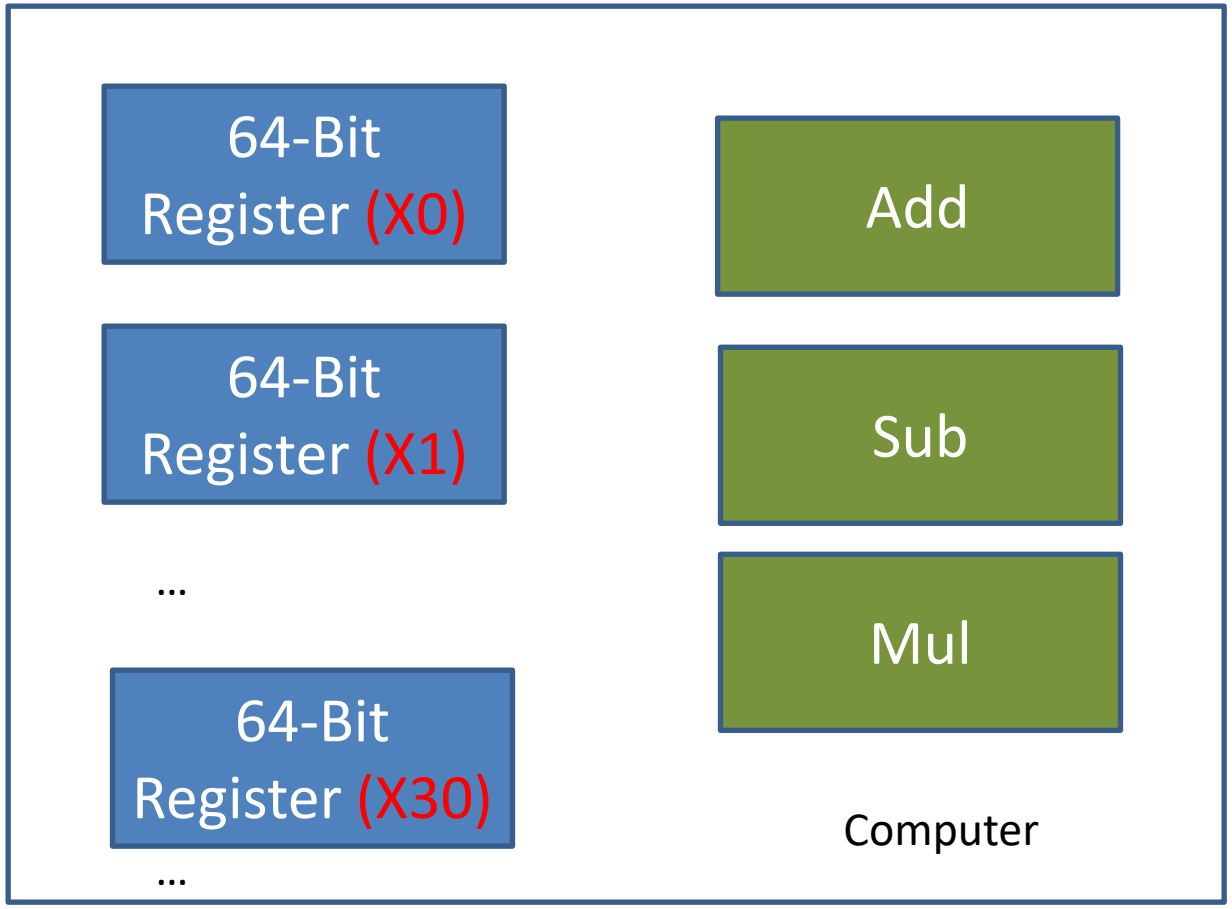
Design Principle 2: Smaller is faster

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.



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…!)



Example – 3 (Again)

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

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

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

Example – 3 (Again)

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

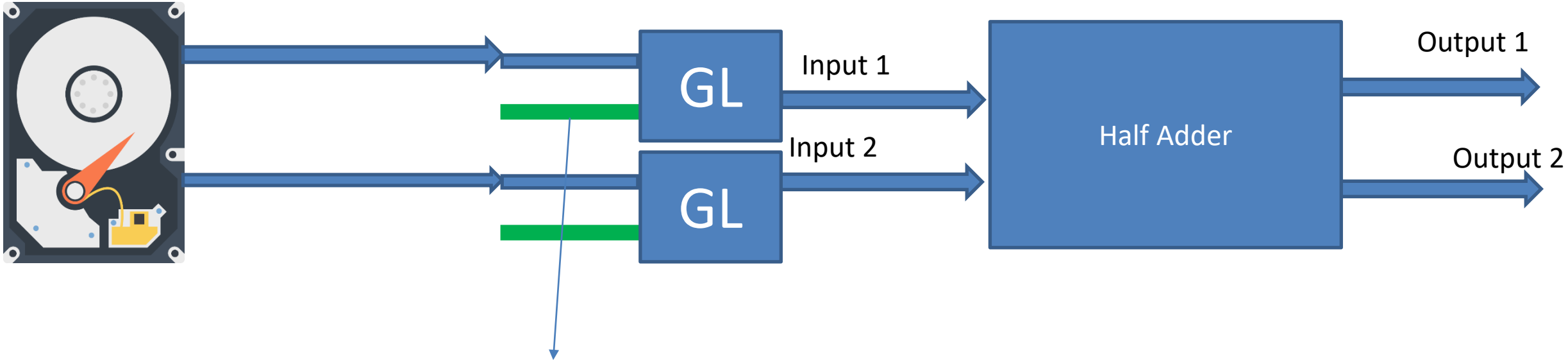
$$f, \ldots, j \text{ store in registers } X19, X20, \ldots, X23$$

$$\text{Two temporary registers are availabe } X9 \text{ \& } X10$$

$$\boxed{\begin{array}{ll} ADD\ X9, X20, X21 & //\ X9 = g + h \\ ADD\ X10, X22, X23 & //\ X10 = i + j \\ SUB\ X19, X9, X10 & //\ f = X9 - X10 \end{array}}$$

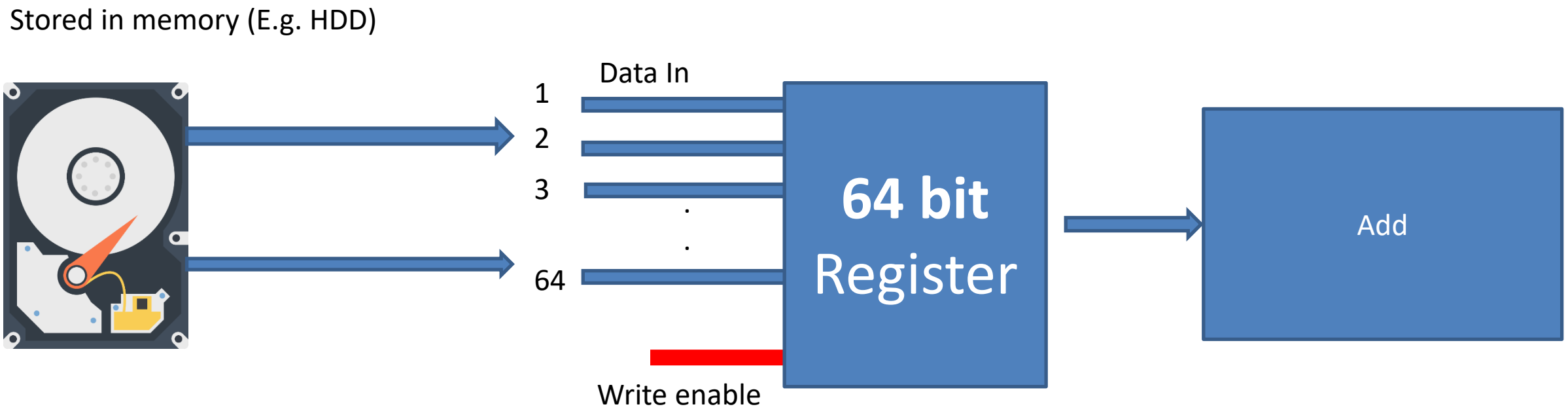
Review: Half-Adder with manual input

Stored in memory (E.g. HDD)

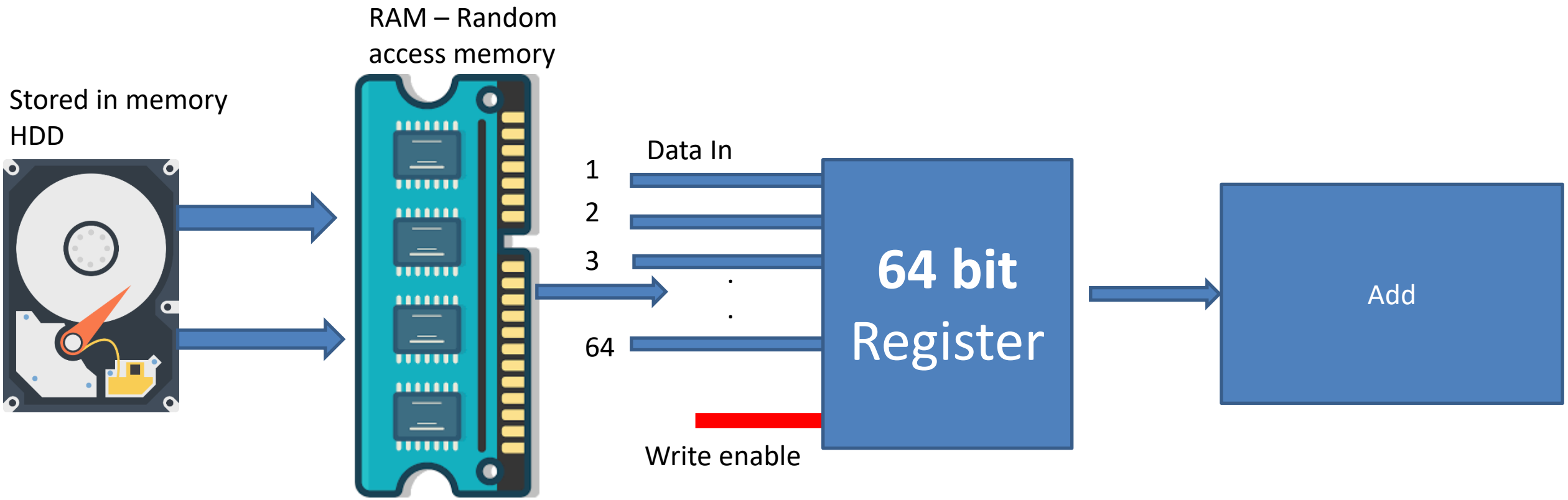


Values updated as
write is enabled

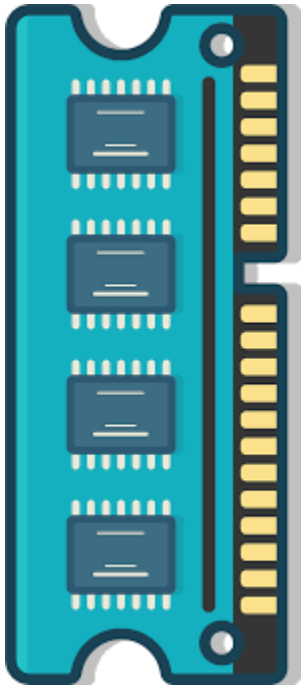
Review: Half-Adder with manual input

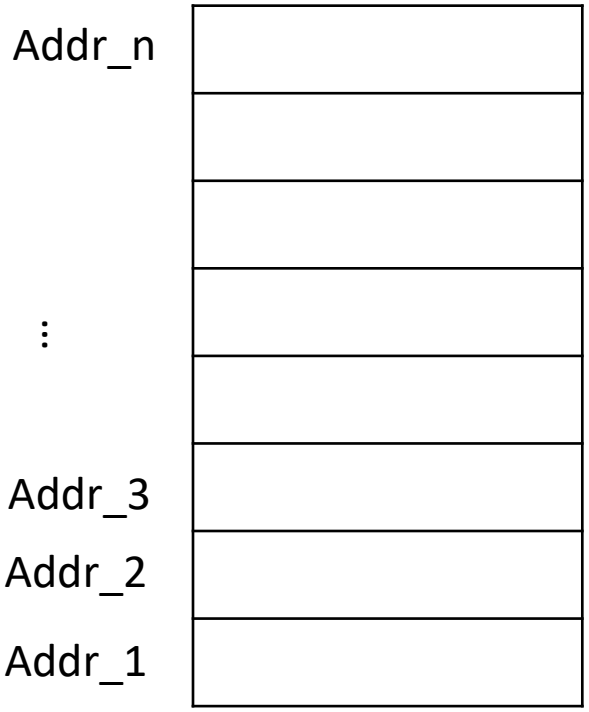


Review: Half-Adder with manual input

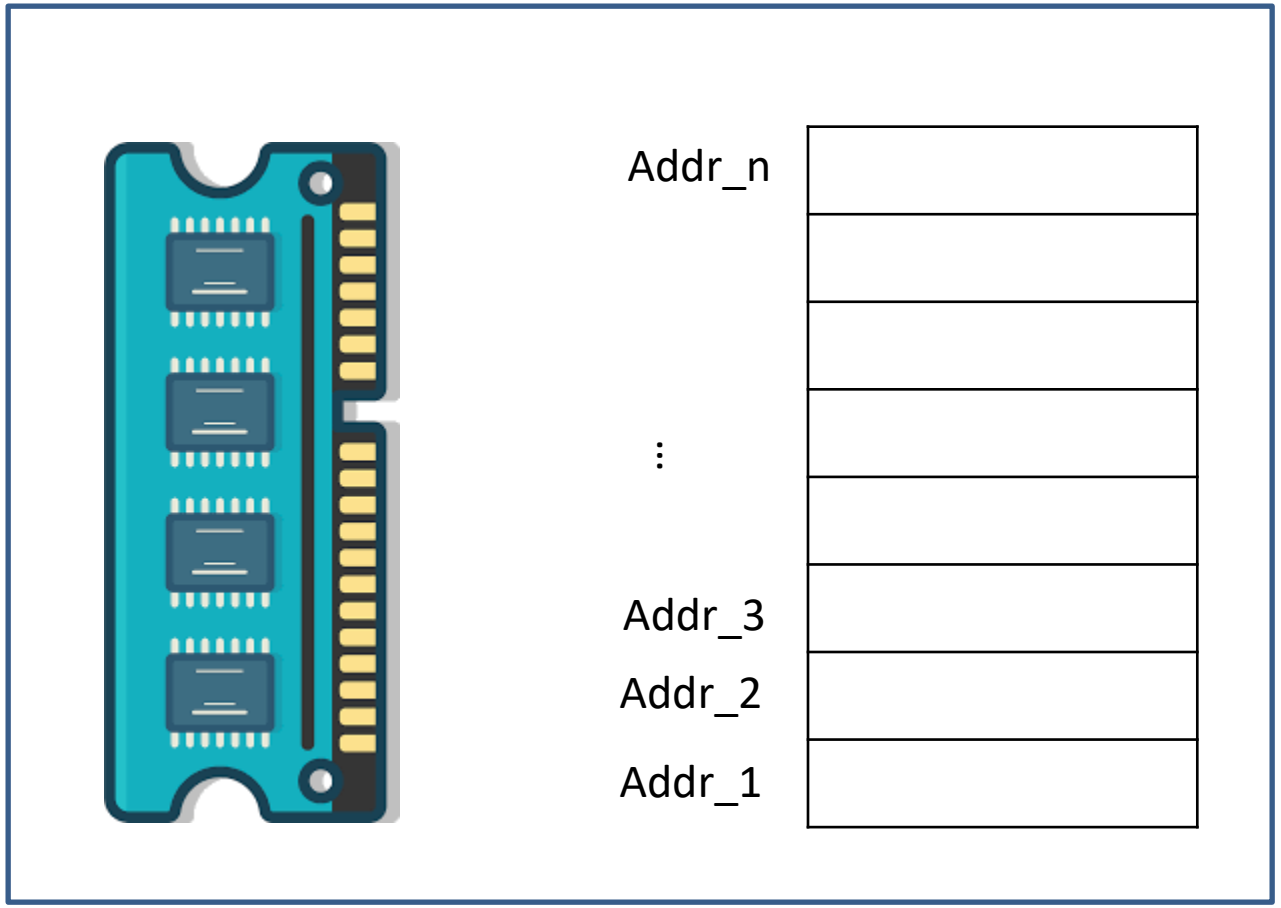


Ram Address

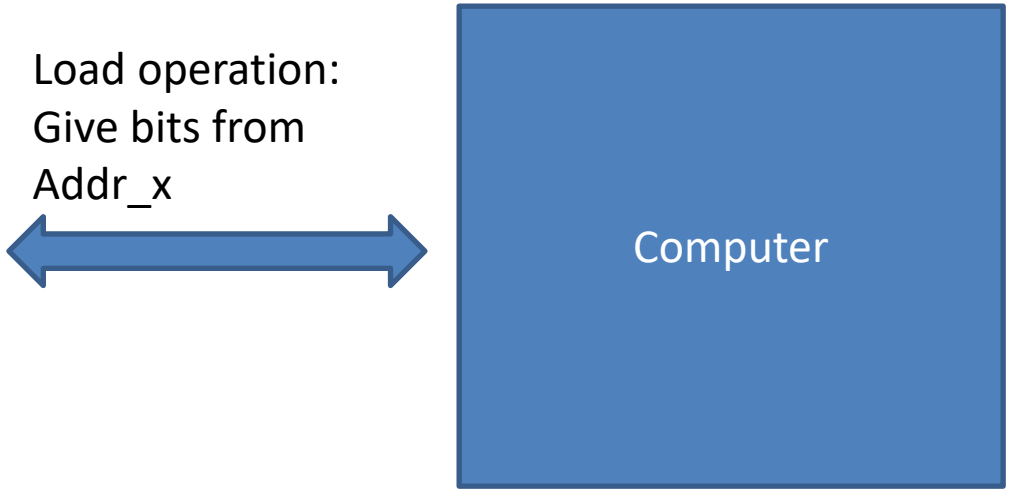




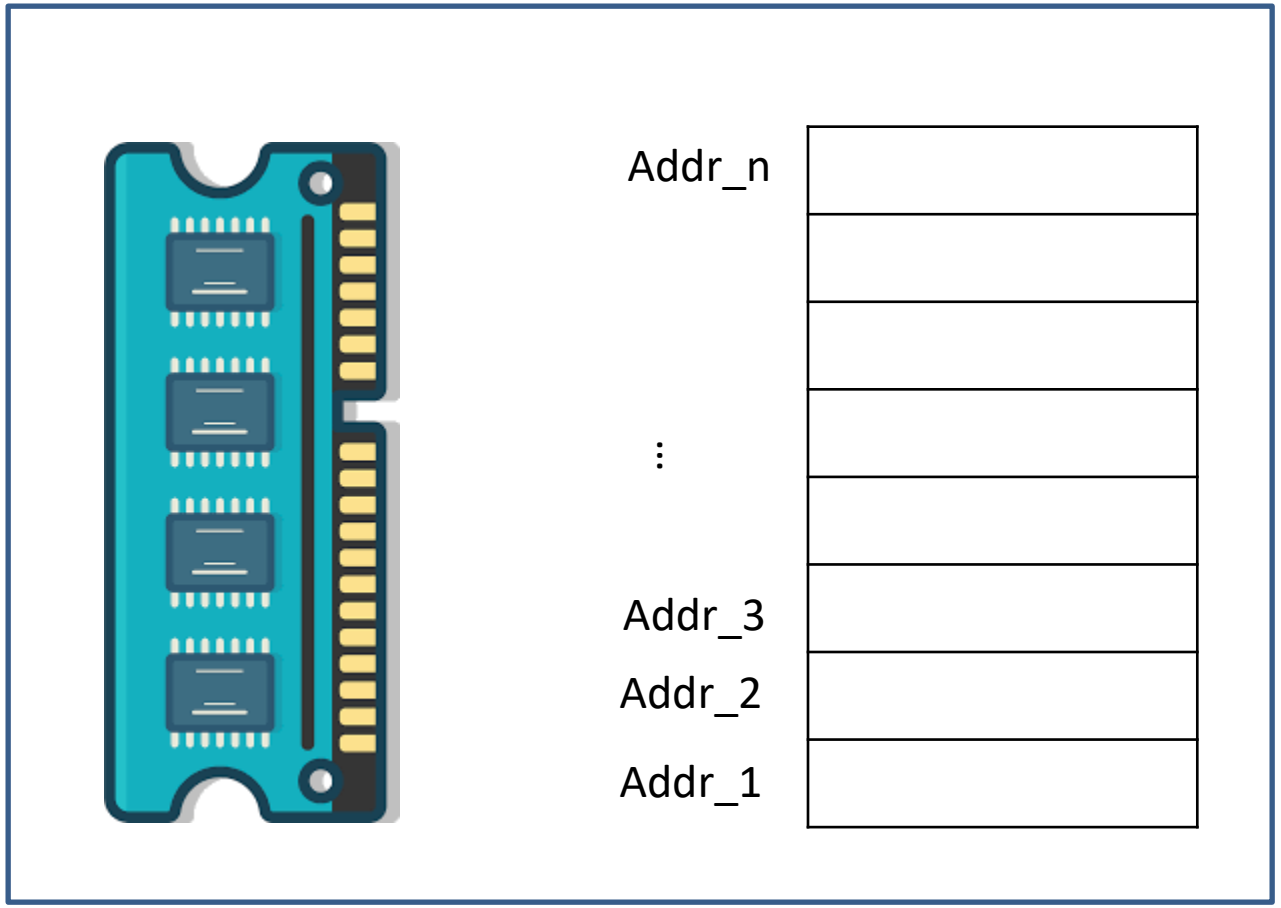
Load Operation



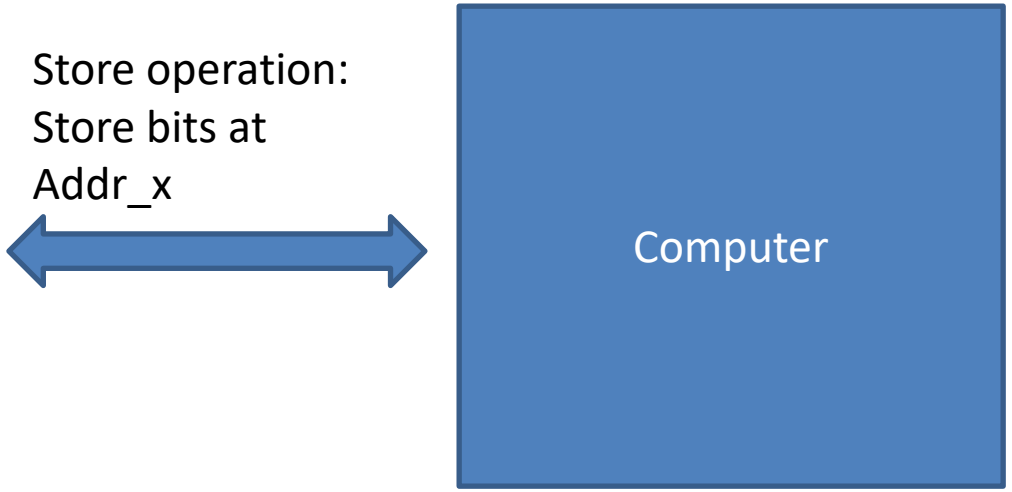
Load operation:
Give bits from
Addr_x



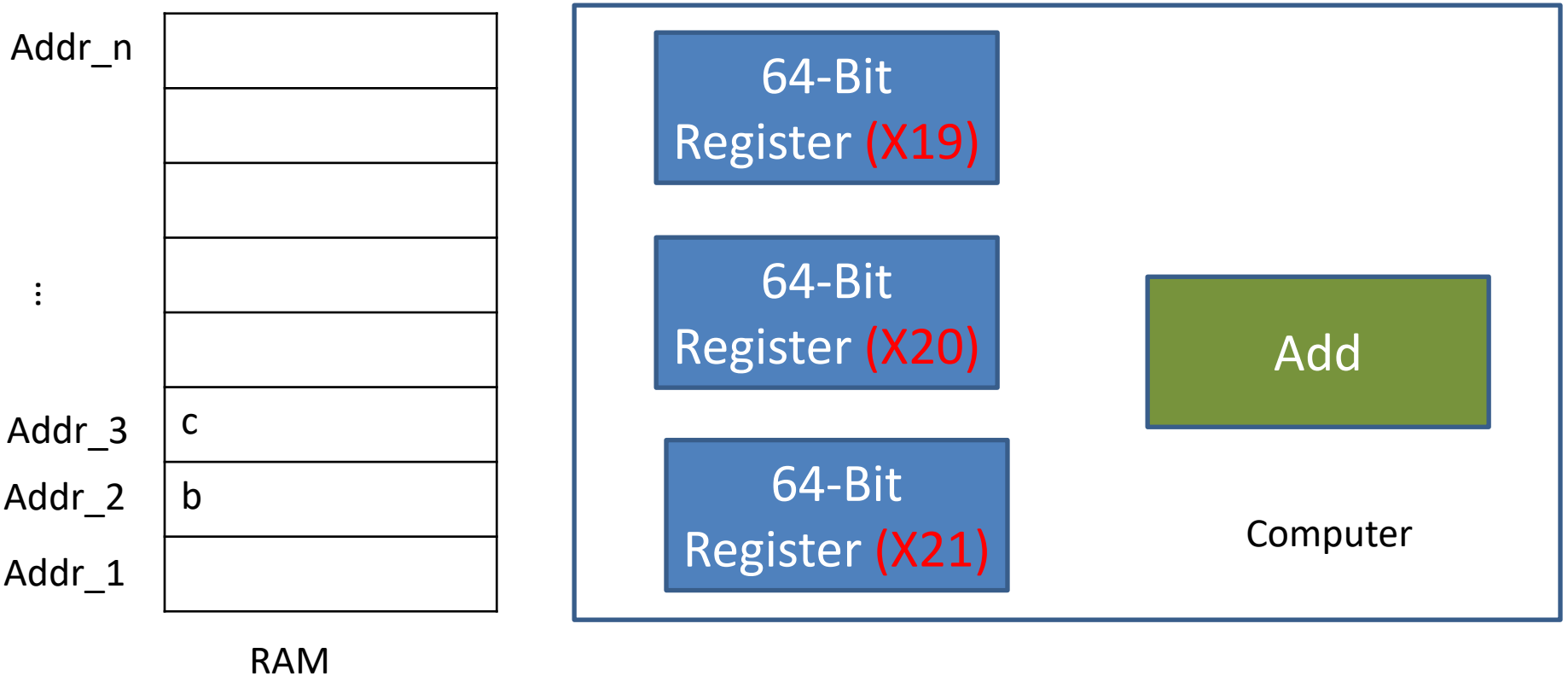
Store Operation



Store operation:
Store bits at
Addr_x

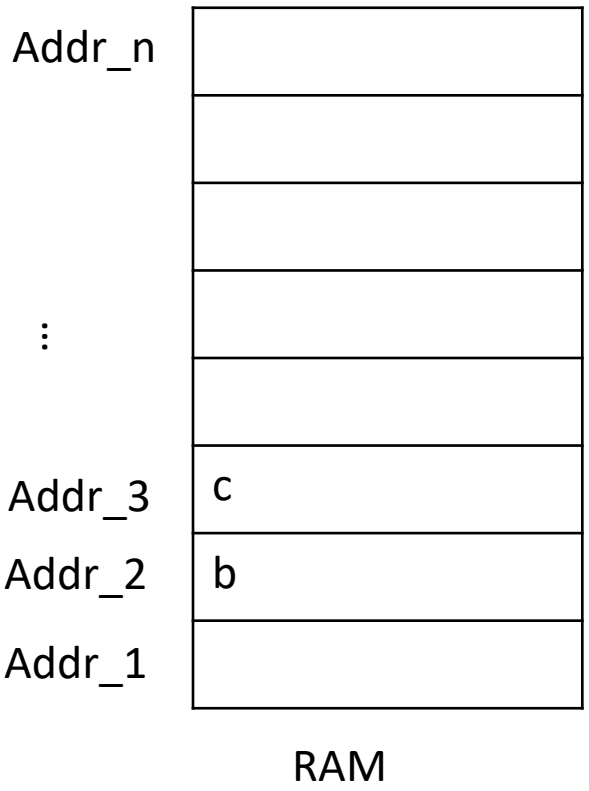


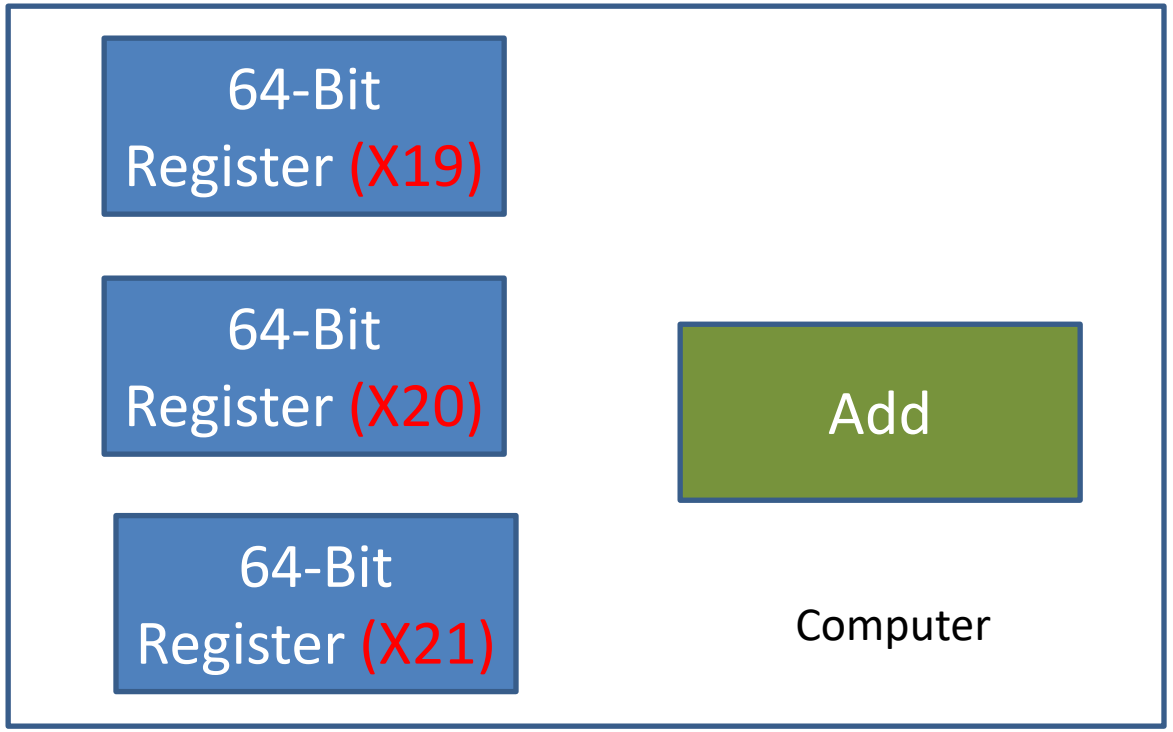
Memory Operand, LOAD



$a = b + c$

Memory Operand , LOAD

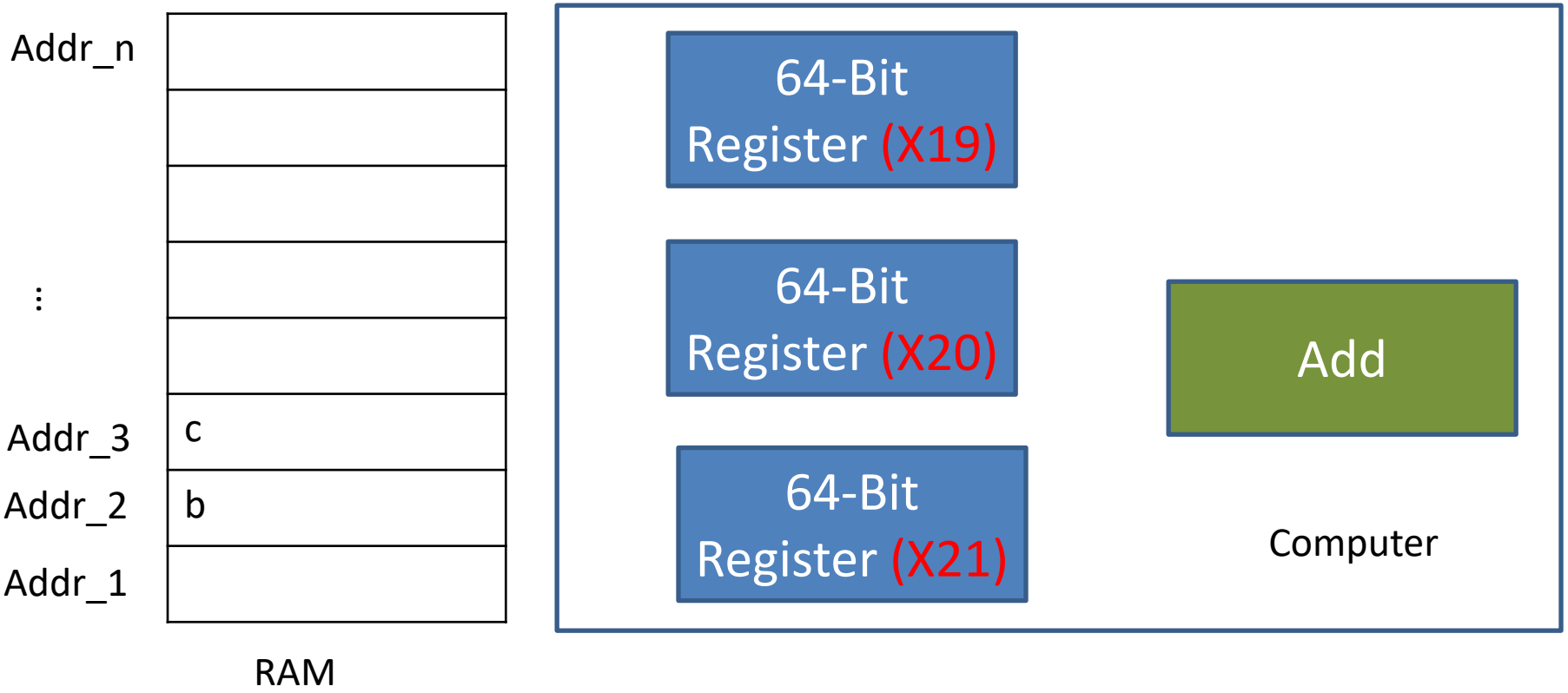




$$a = b + c$$

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

Memory Operand , LOAD

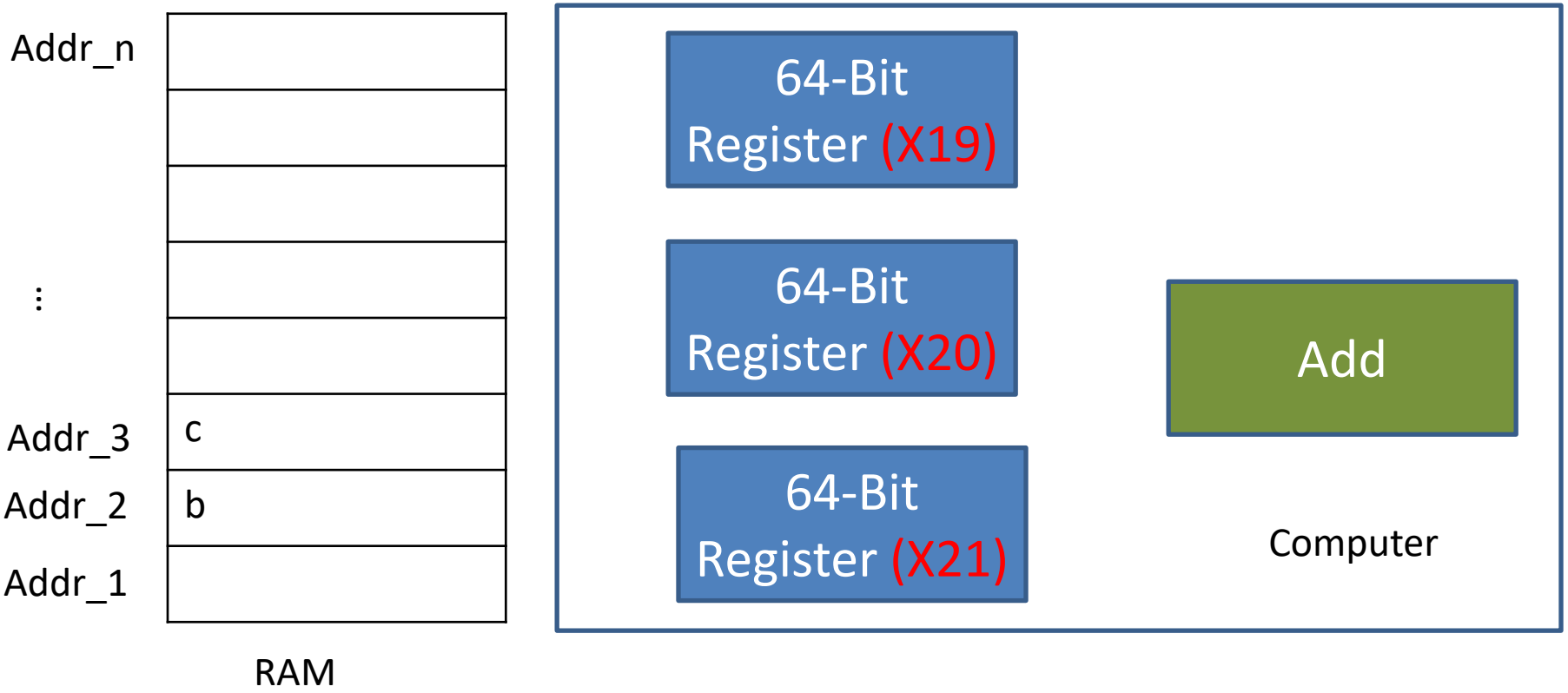


$$a = b + c$$

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 Operand , LOAD



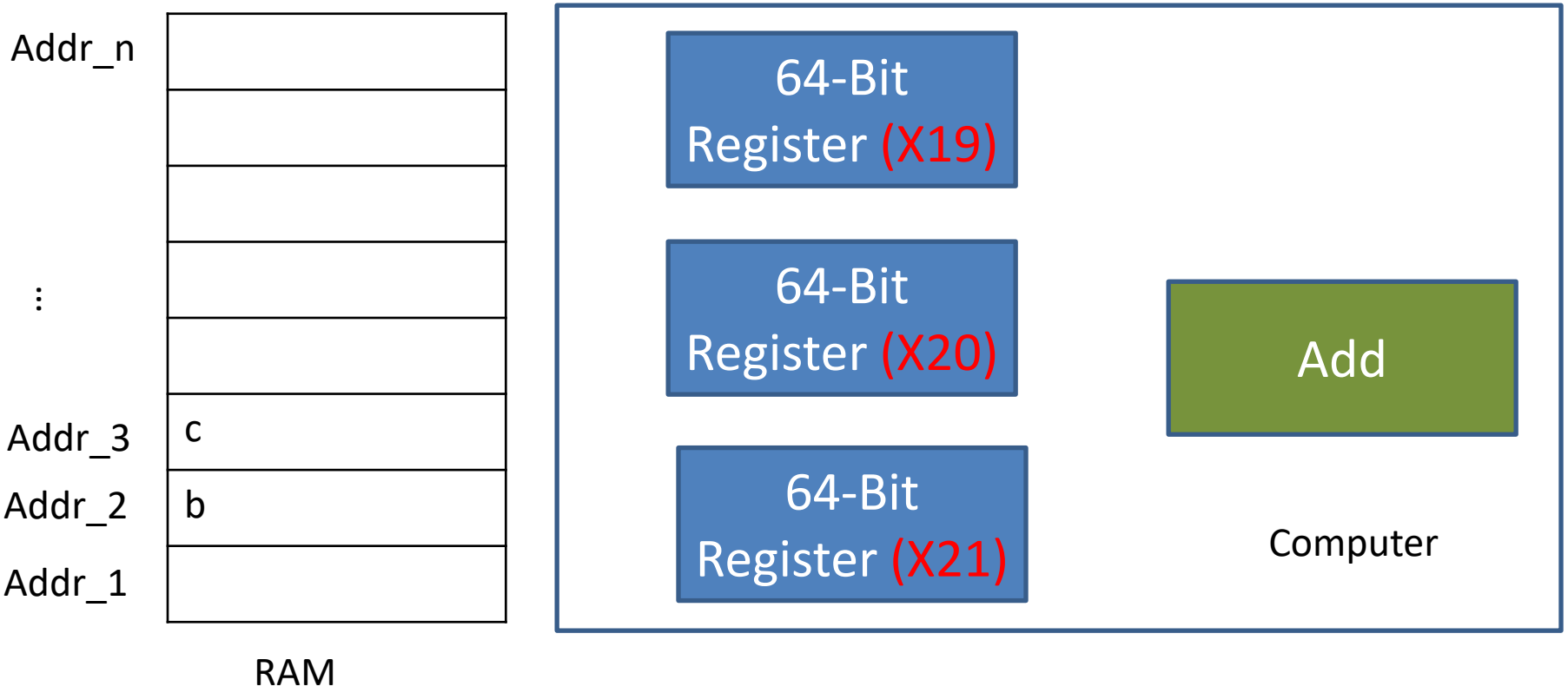
$$a = b + c$$

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.

Memory Operand , LOAD

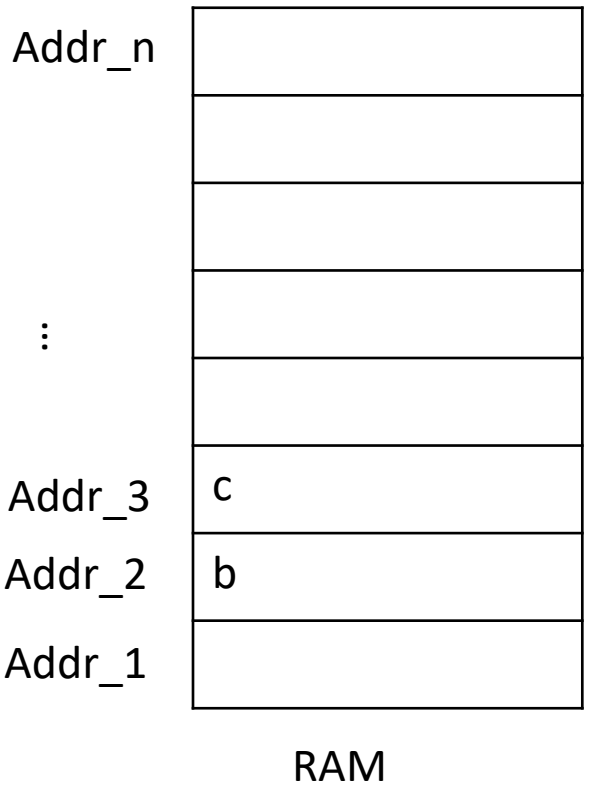


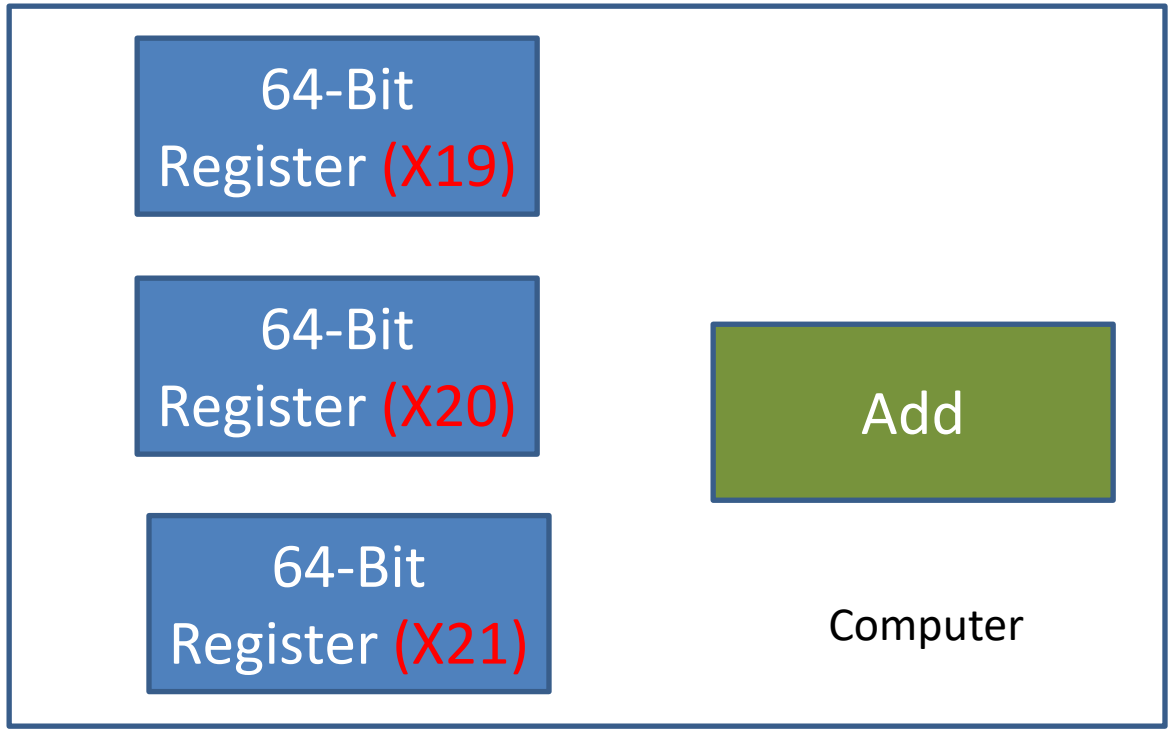
$$a = b + c$$

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

Memory Operand , LOAD





$$a = b + c$$

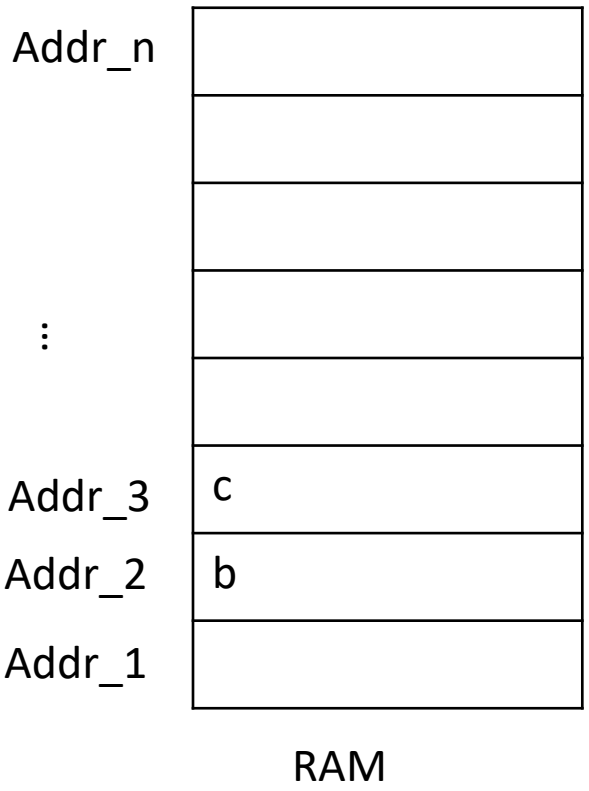
Load Addr_3 (c) to register X19
Load Addr_2 (b) to register X20
ADD X21, X19, X20

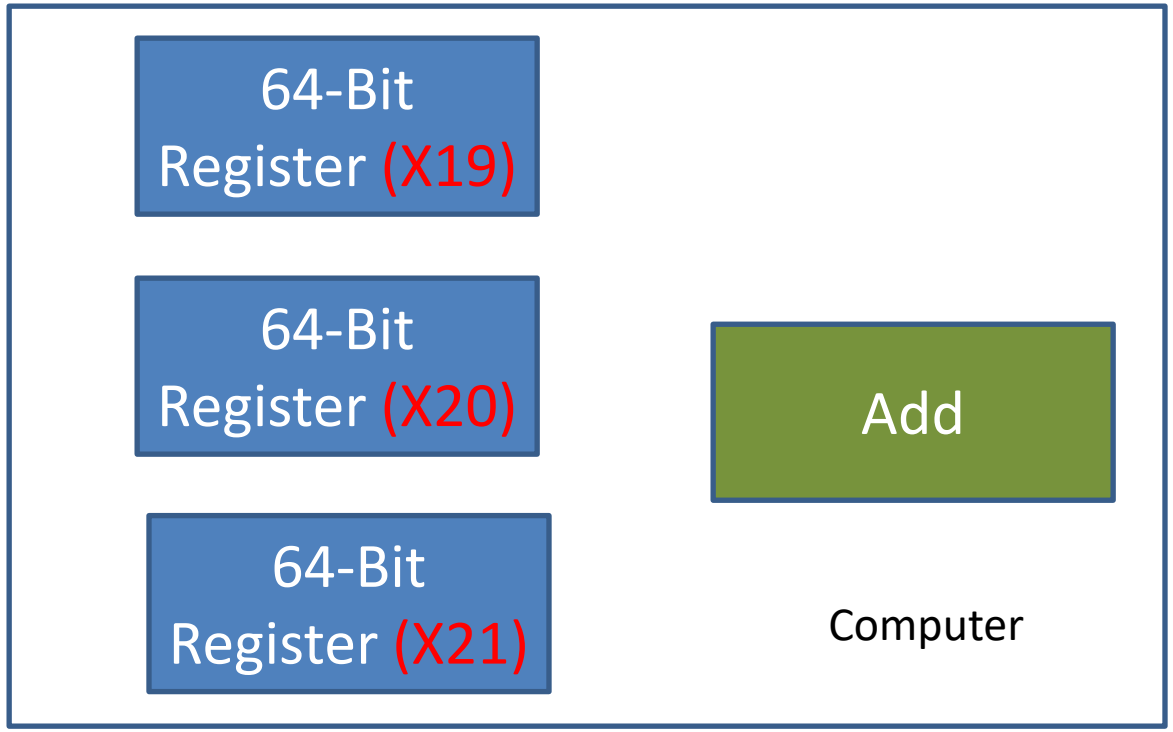
Lets assume memory address is stored in X22

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

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

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

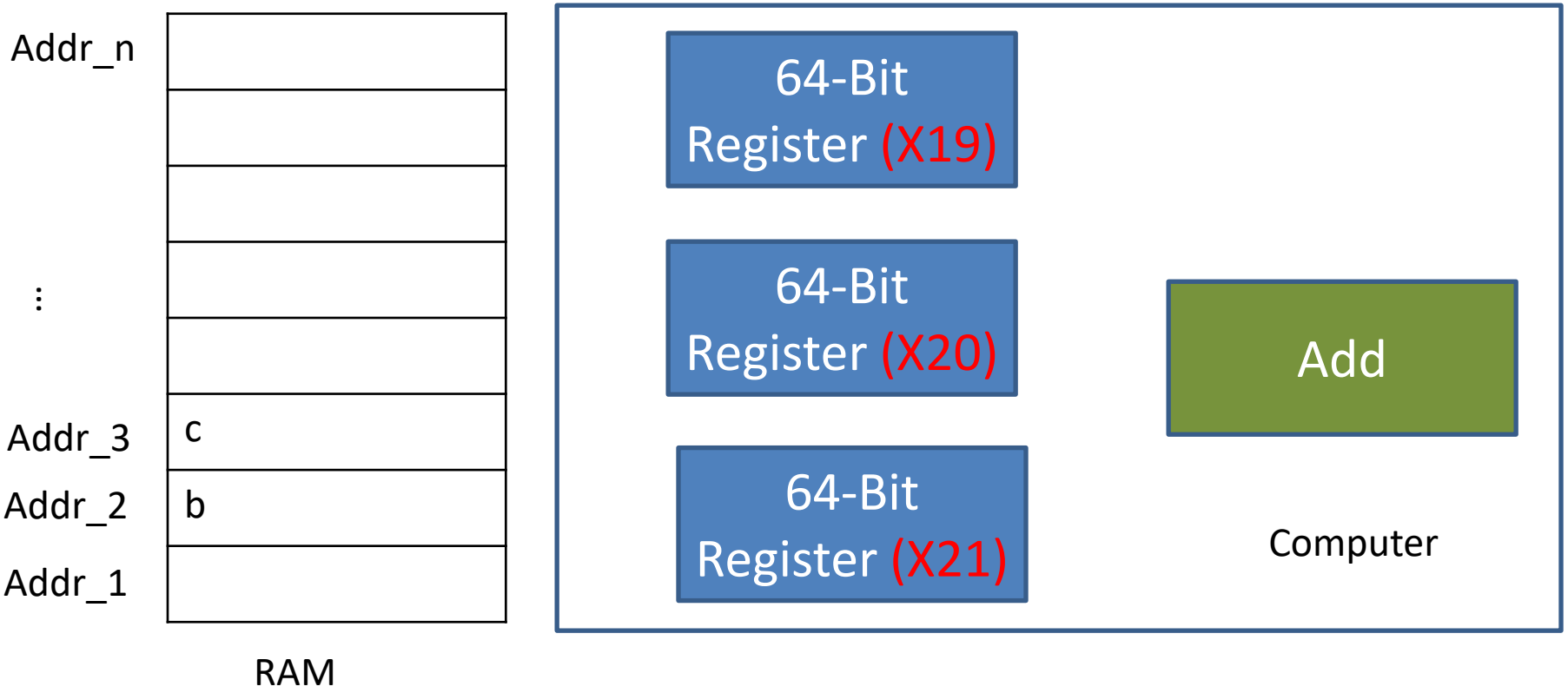
The **LEGv8 instruction** to:
Load Addr_2 (a) to register X19

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

Destination register

Ram address

Memory Operand , LOAD



$$a = b + c$$

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

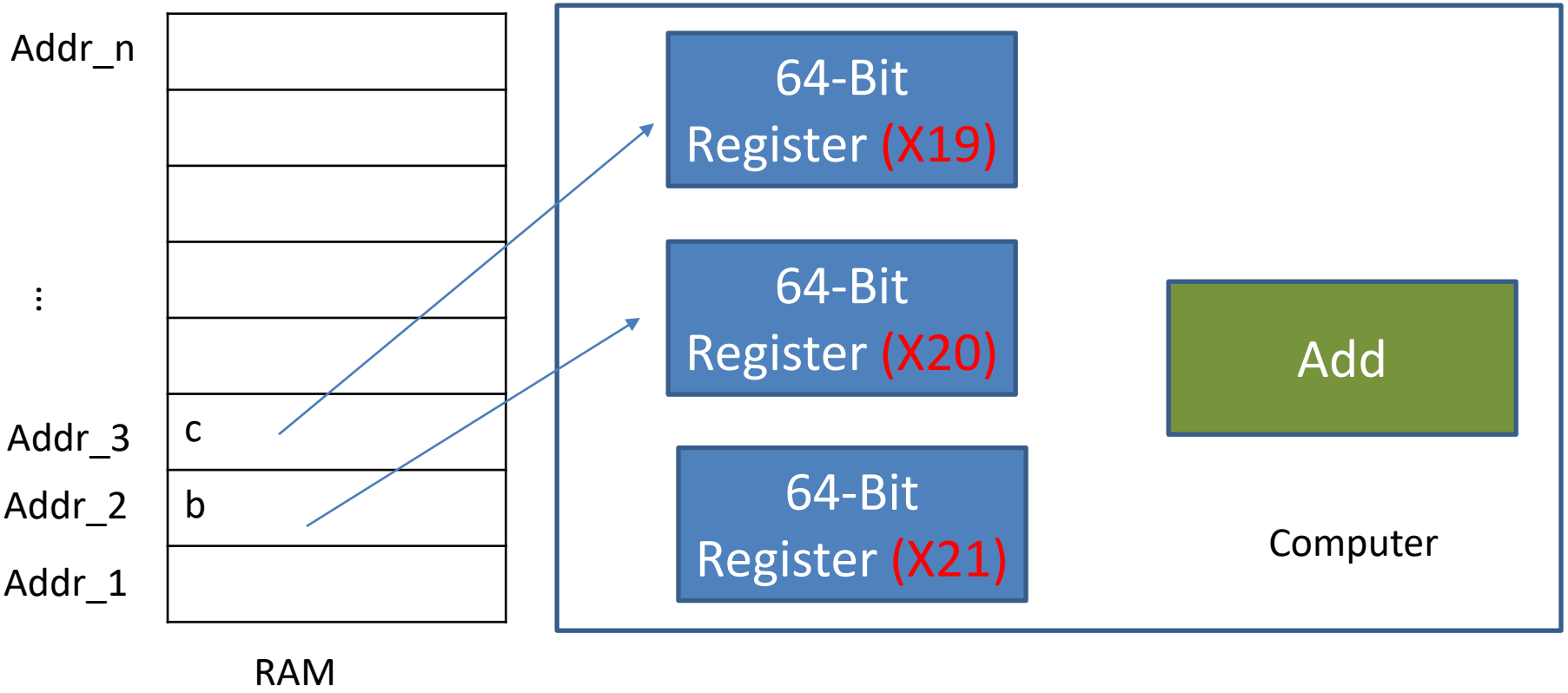
Lets assume memory address Addr_2 is stored in X22

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

$$LDUR\ X19, [X22, \#\boldsymbol{const}]$$

Constant, more on this later! Lets use **0** for now

Memory Operand , LOAD



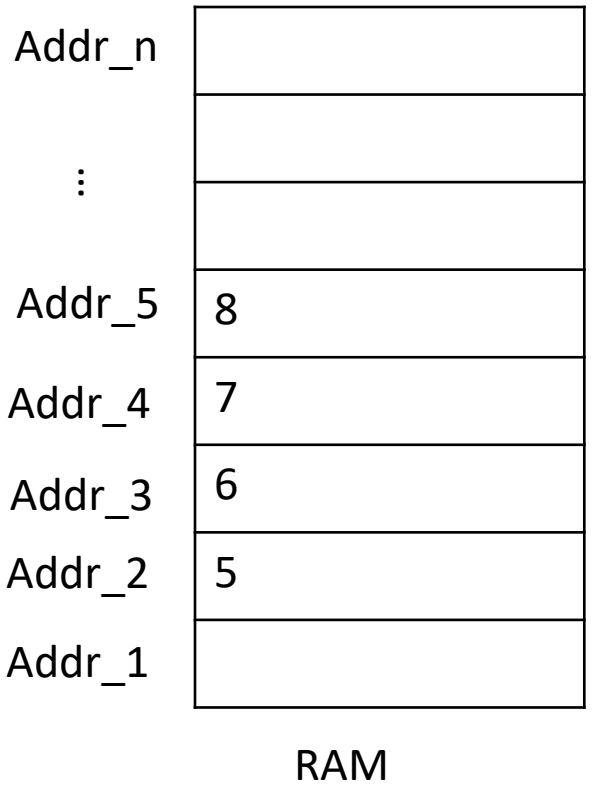
$$a = b + c$$

Assuming Addr_3 is store in X22 and Addr_2 is store in X23

Assembly code **(LEGv8 instruction)**

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

Arrays in RAM



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

1. Arrays are stored in contiguous memory

Let a start from Addr_2

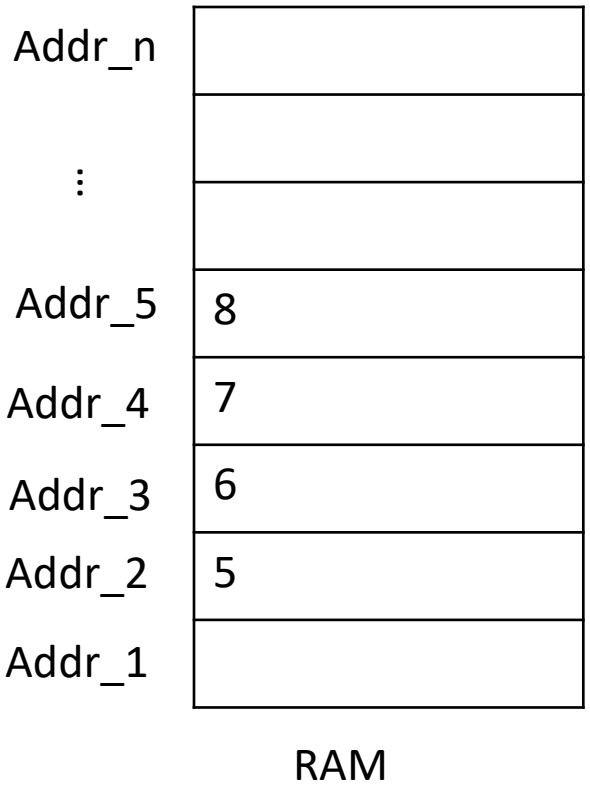
a[0] ➔Addr_2

a[1] ➔ Addr_3

a[2] ➔Addr_4

a[3] ➔Addr_5

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

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

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

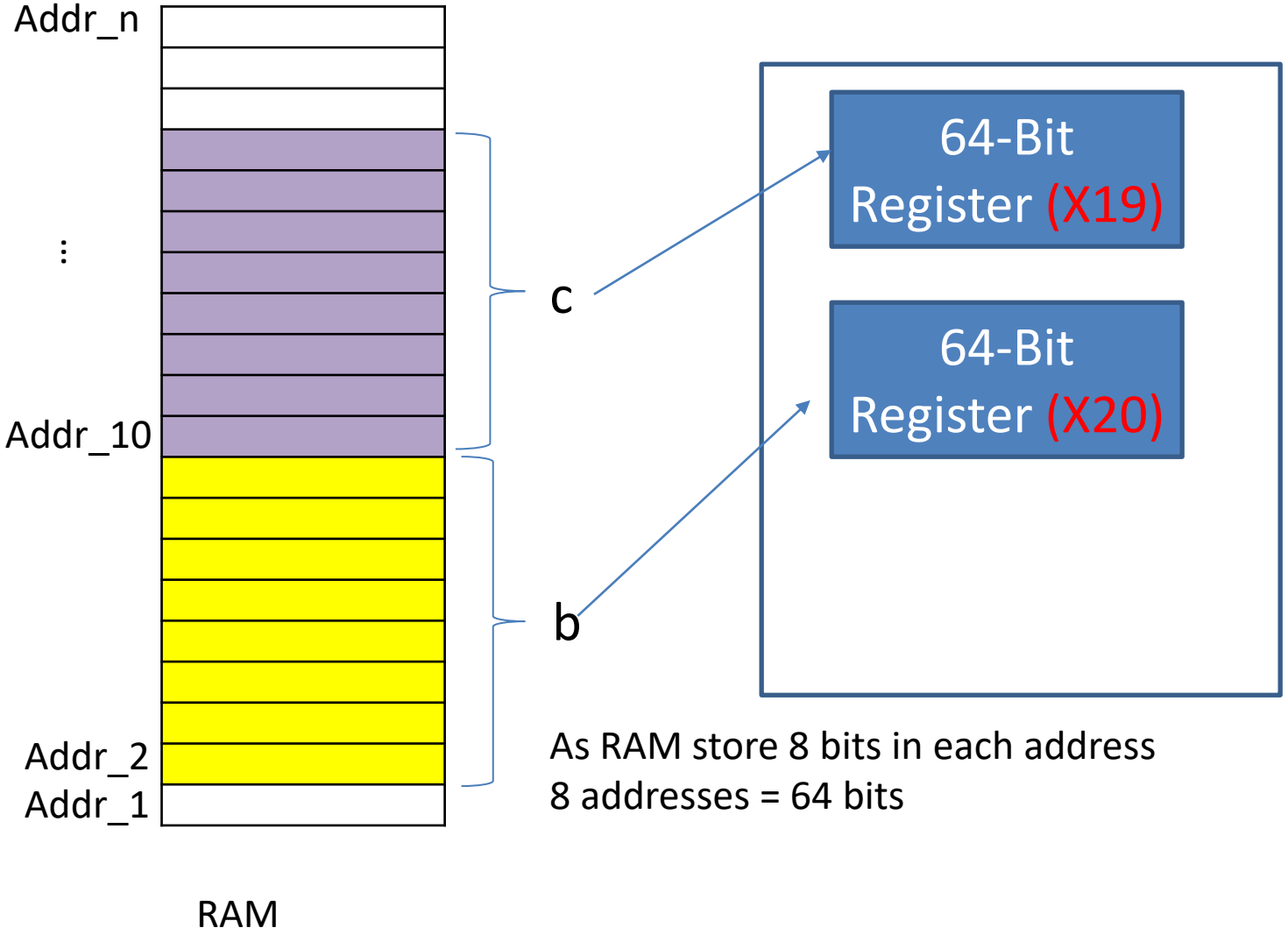
RAMS and Byte Addresses

Addr_n	
⋮	
Addr_5	
Addr_4	
Addr_3	10011011
Addr_2	10110011
Addr_1	10010011

RAM

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.

Memory Operand , LOAD



As RAM store 8 bits in each address
8 addresses = 64 bits

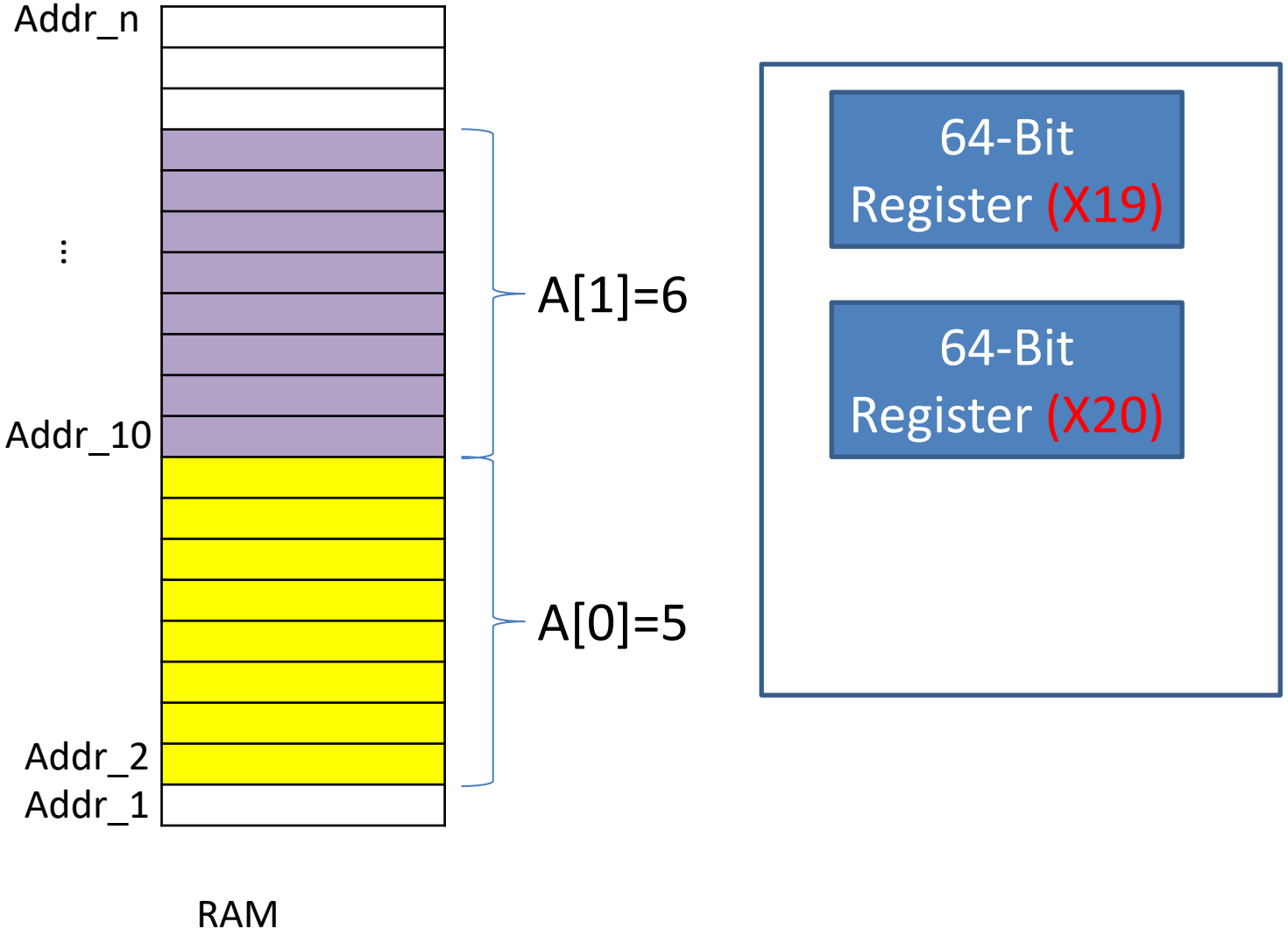
LEGv8 Registers are 64 bit.
Require 8 cells in ram to store value

Assuming Addr_10 is store in X22
and Addr_2 is store in X23
Assembly code **(LEGv8 instruction)**

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

Designed to automatically load 8 contiguous cells from ram into register

Memory Operand , LOAD

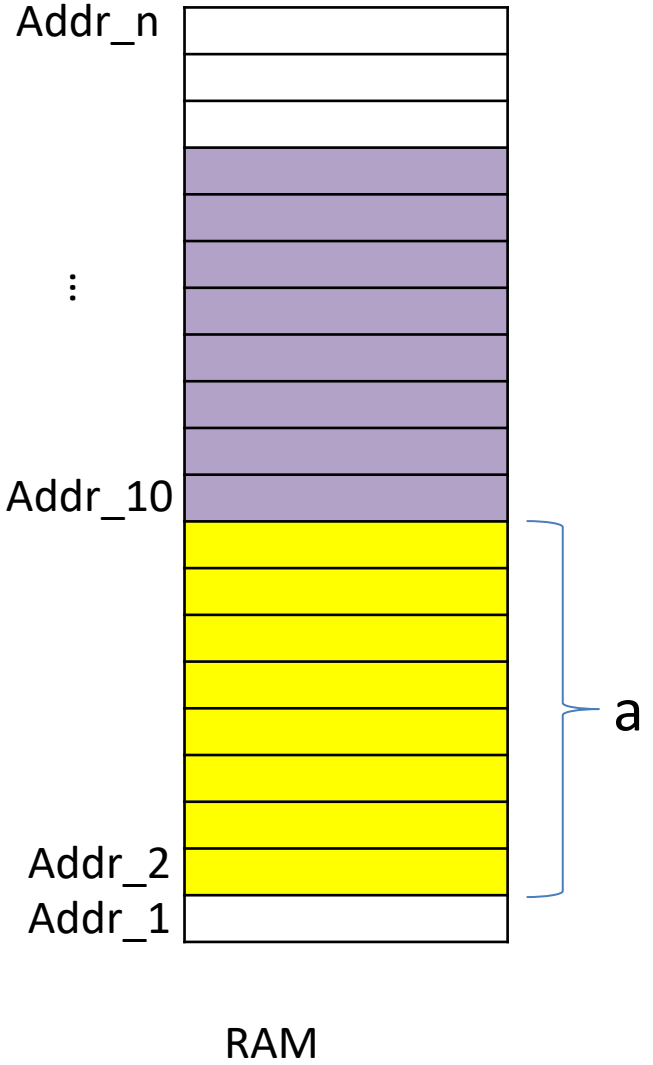


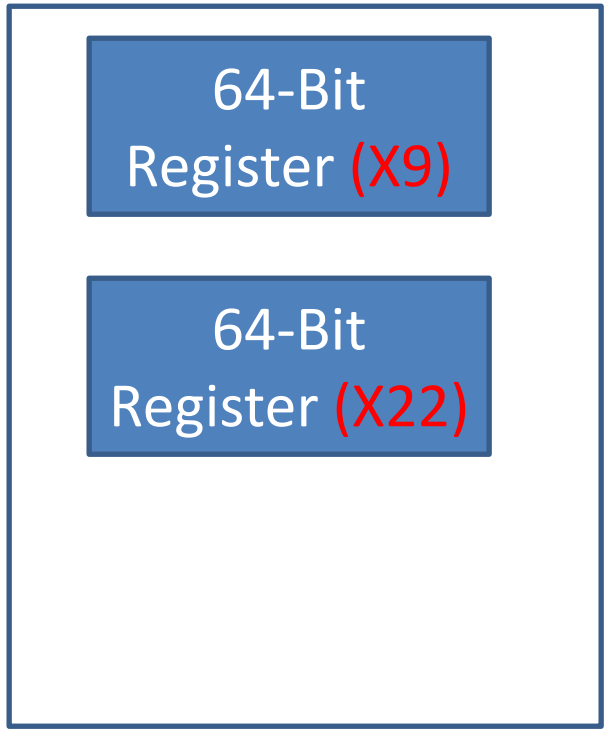
$int\ a[4] = \{5, 6, 7, 8\};$

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?

Memory Operand , LOAD



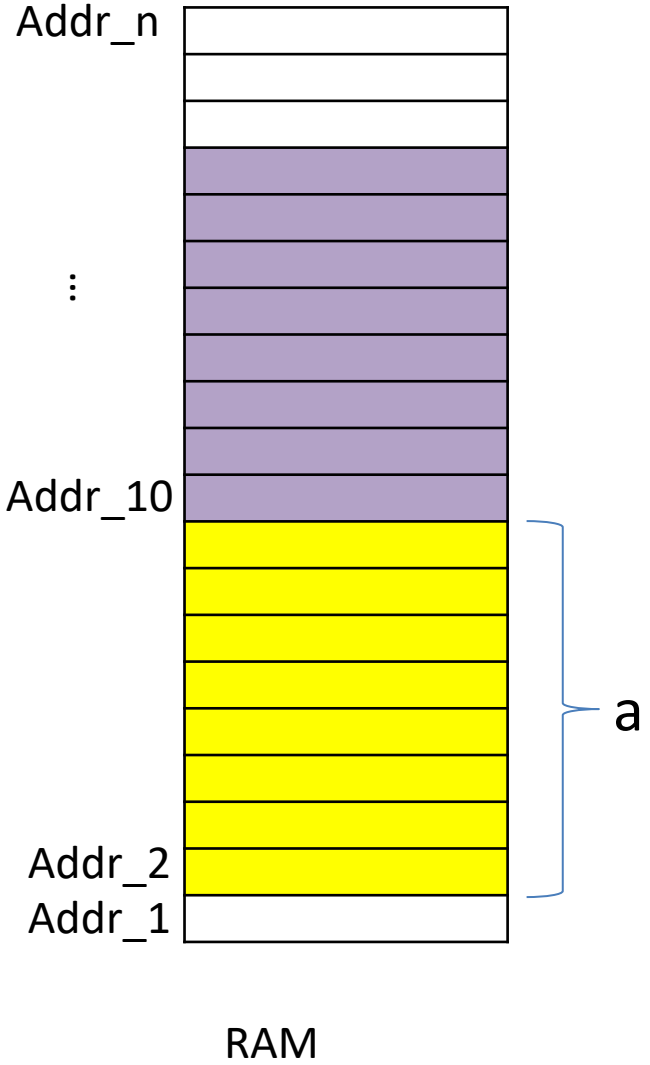


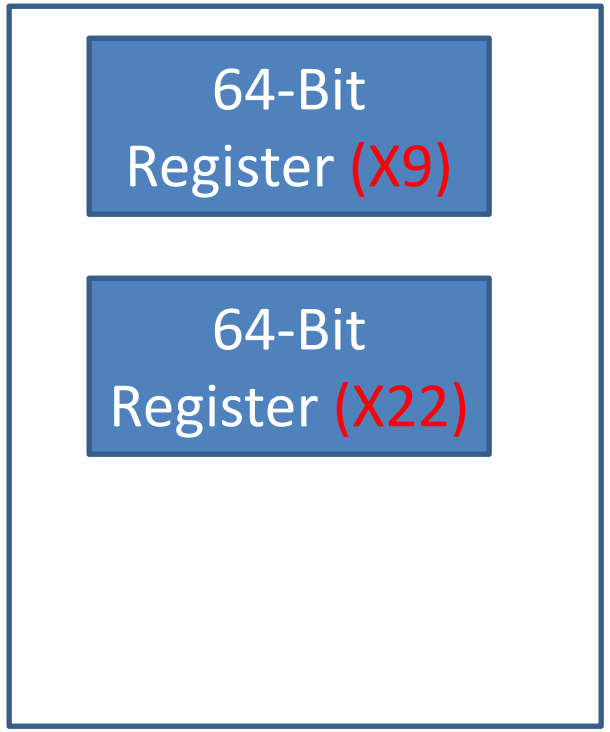
$$int\ a[4] = \{5, 6, 7, 8\};$$

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?

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

Memory Operand , LOAD





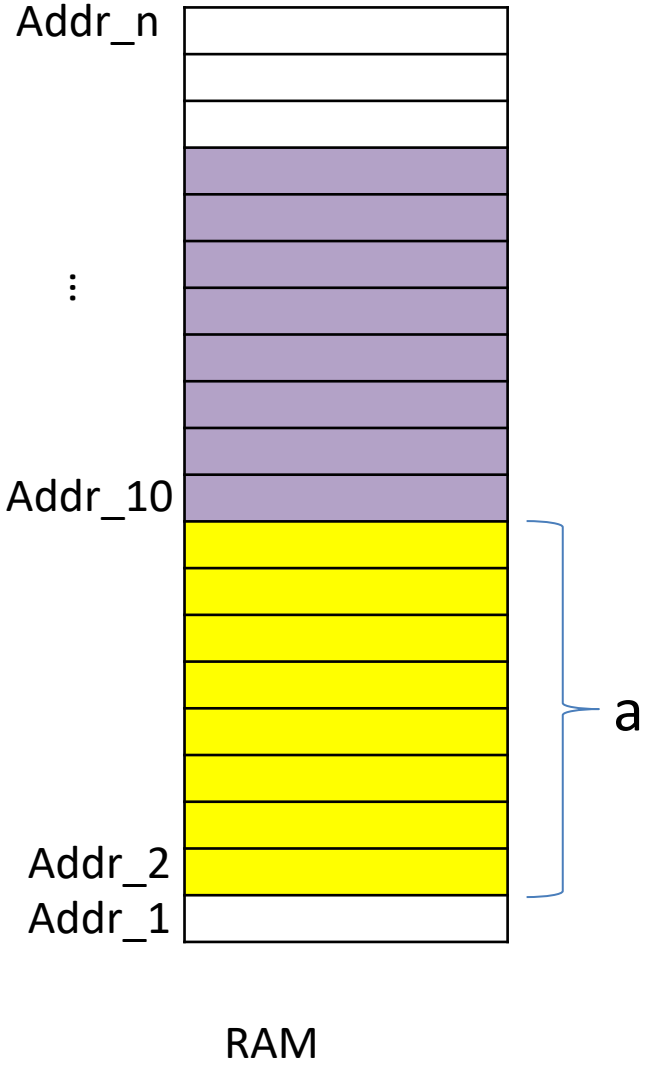
$$int\ a[4] = \{5, 6, 7, 8\};$$

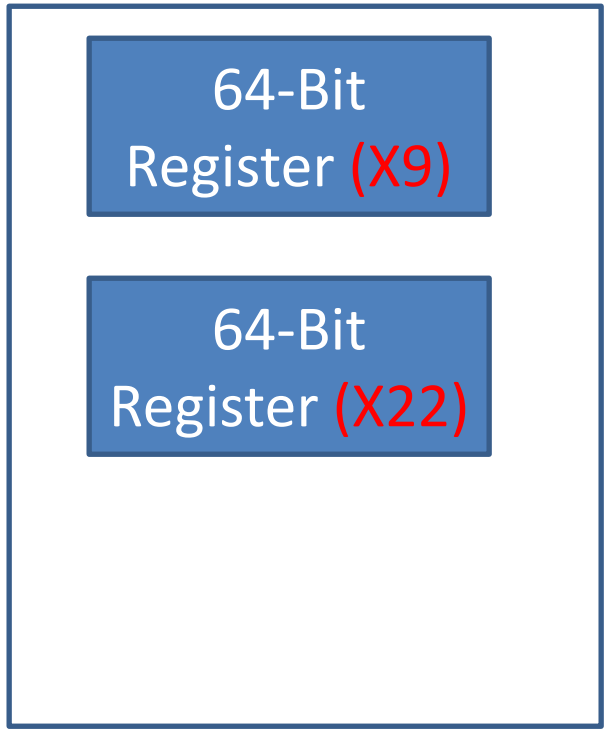
Let start address be Addr_2
Let Addr_2 be stored in register X22
What is the LEGv8 instruction to load a[0] into register X9?

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

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

Memory Operand , LOAD





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

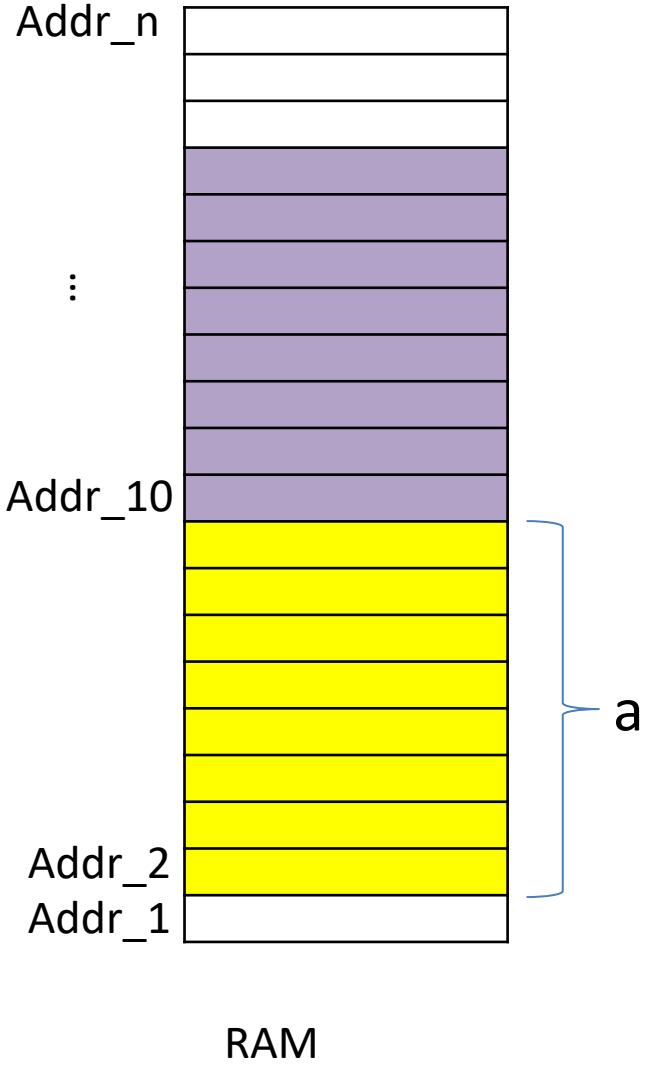
Let start address be Addr_2
Let Addr_2 be stored in register X22
What is the LEGv8 instruction to load a[0] into register X9?

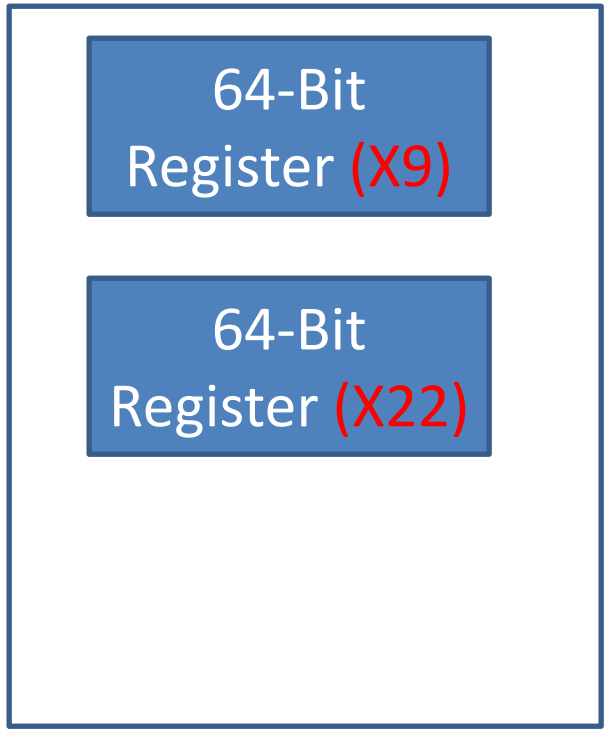
$$LDUR\ X9, [X22, \#\mathbf{0}]$$

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

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

Memory Operand , LOAD





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

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?

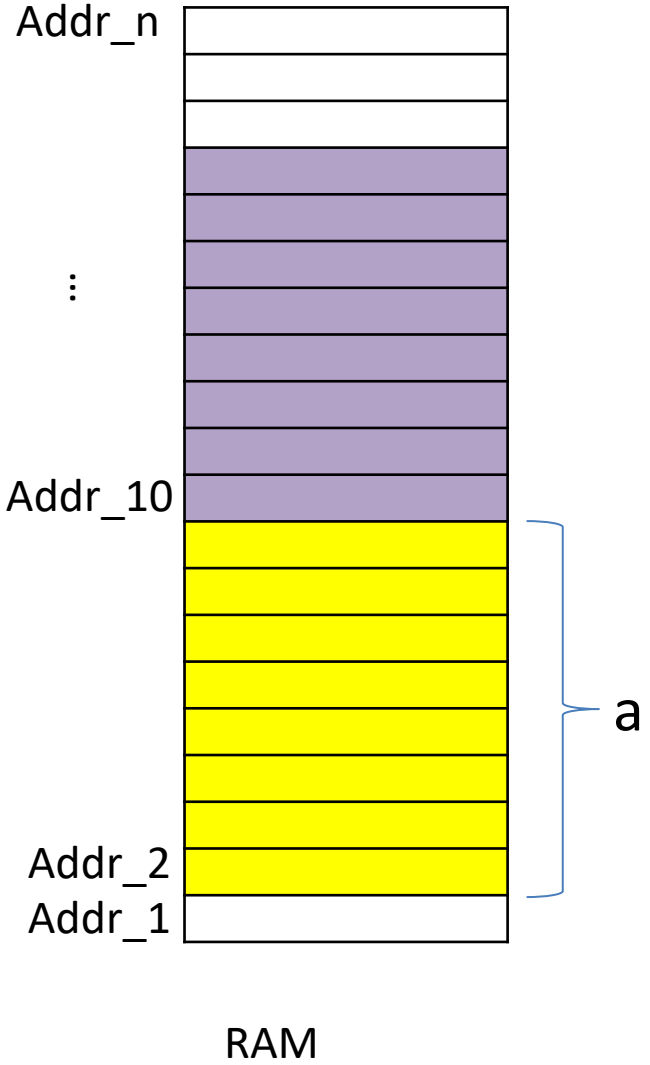
$$LDUR\ X9, [X22, \#\mathbf{0}]$$

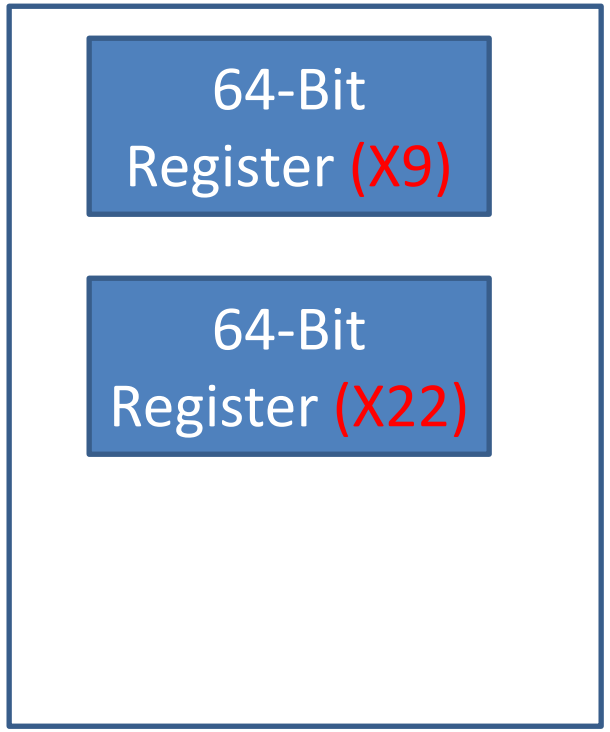
What is the LEGv8 instruction to load a[1] into register X9?

$$LDUR\ X9, [X22, \#\mathbf{8}]$$

What is the LEGv8 instruction to load a[4] into register X9?

Memory Operand , LOAD





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

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?

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

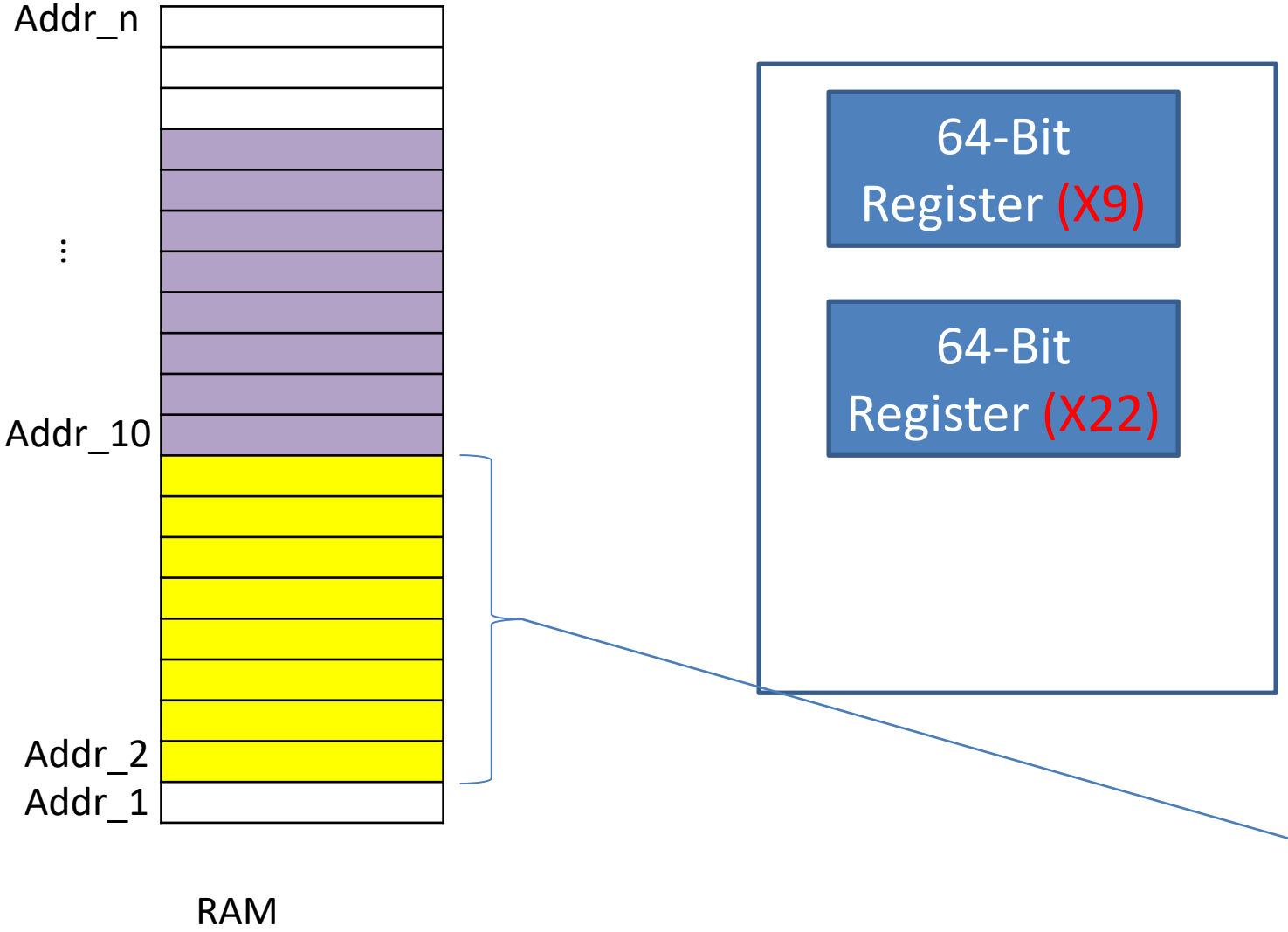
What is the LEGv8 instruction to load a[1] into register X9?

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

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

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

Memory Operand , LOAD



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

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

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:

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]
```

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
```

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]
```

Chapter 1 – Performance review

Performance Review

Our favorite program runs in 10 seconds on computer A, which has a 2 GHz clock. We are trying to help a computer designer build a computer, B, which will run this program in 6 seconds. The designer has determined that a substantial increase in the clock rate is possible, but this increase will affect the rest of the CPU design, causing computer B to require 1.2 times as many clock cycles as computer A for this program. What clock rate should we tell the designer to target?

Variables

Variables
Clock cycle
Clock cycle count
Clock cycle time
Clock period
Clock rate
Clock cycles per instruction
Instruction count
Execution Time

Variables

Variables
Clock cycle
Clock cycle count
Clock cycle time
Clock period
Clock rate
Clock cycles per instruction
Instruction count
Execution Time

Too many clocks!!!!

Variables

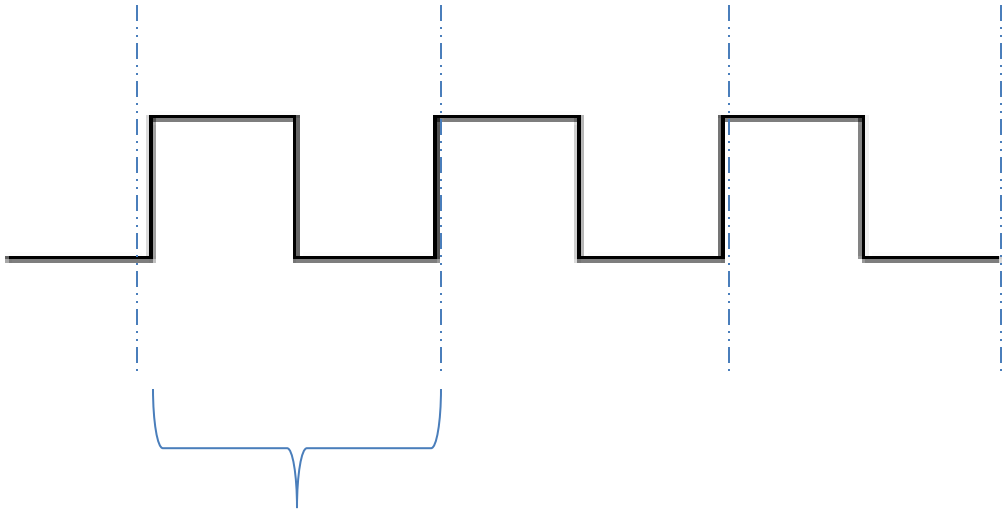
Variables
~~Clock~~ cycle
~~Clock~~ cycle count
~~Clock~~ cycle time
~~Clock~~ period
~~Clock~~ rate
~~Clock~~ cycles per instruction
Instruction count
Execution Time

Too many clocks!!!!

Variables

Variables
Cycle
cycle count
Cycle time
Period
Rate
Cycles per instruction
Instruction count
Execution Time

- 1 execution performed by the computer

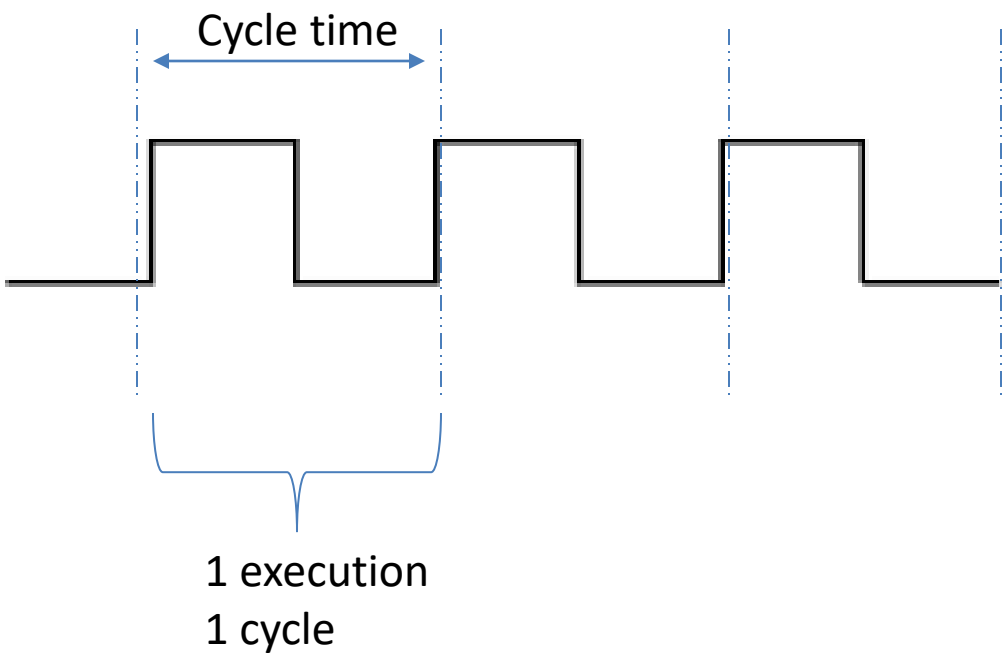


1 execution
1 cycle

Variables

Variables
Cycle
Cycle count
Cycle time
Period
Rate
Cycles per instruction
Instruction count
Execution Time

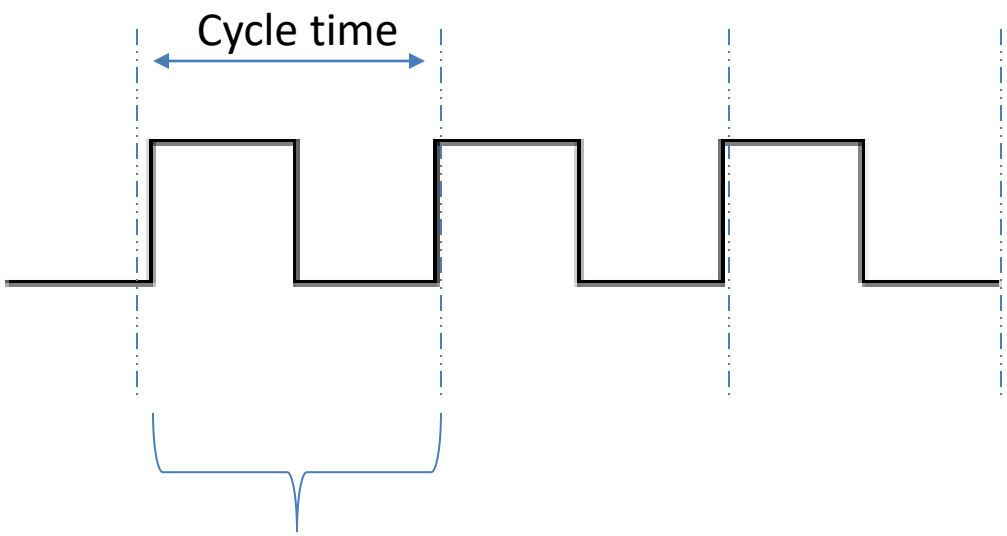
- 1 execution performed by the computer
- Time for one cycle



Variables

Variables
Cycle
Cycle count
Cycle time
Period
Rate
Cycles per instruction
Instruction count
Execution Time

- 1 execution performed by the computer
- Cycles needed to run a program
- Time for one cycle



1 execution
1 cycle

Performance

- To define performance of a computer A
- Run a program and compute the time to complete (execution time)
- Less the time better is the performance

$$performance_A \propto \frac{1}{(execution\ time)}$$

Performance

- To define performance of a computer A
- Run a program and compute the time to complete (execution time)
- Less the time better is the performance

$$performance_A \propto \frac{1}{\boxed{(execution\ time)}}$$

Variables

Variables
Cycle
Cycle count
Cycle time
Period
Rate
Cycles per instruction
Instruction count
Execution Time

- 1 execution performed by the computer
- Cycles needed to run a program
- Time for one cycle

If a program needs $5X10^9$ cycles, and the cycle time $100\ picoseconds$ on computer A.
Then the execution time ?

Variables

Variables
Cycle
Cycle count
Cycle time
Period
Rate
Cycles per instruction
Instruction count
Execution Time

- 1 execution performed by the computer
- Cycles needed to run a program
- Time for one cycle

If a program needs $\mathbf{5X10^9}$ **cycles**, and the cycle time $100\ picoseconds$ on computer A.
Then the execution time ?

Variables

Variables
Cycle
Cycle count
Cycle time
Period
Rate
Cycles per instruction
Instruction count
Execution Time

- 1 execution performed by the computer
- Cycles needed to run a program
- Time for one cycle

If a program needs $5X10^9$ cycles, and the cycle time $100\ picoseconds$ on computer A.
Then the execution time ?

$$Execution\ time = cycle\_count\ X\ cycle\_time$$

Variables

Variables
Cycle
Cycle count
Cycle time
Period
Rate
Cycles per instruction
Instruction count
Execution Time

- 1 execution performed by the computer
- Cycles needed to run a program
- Time for one cycle

If a program needs $5X10^9$ cycles, and the cycle time $100\ picoseconds$ on computer A.
Then the execution time ?

$$Execution\ time = cycle\_count\ X\ cycle\_time$$
$$= (5X10^9)X\ (100\ X\ 10^{-12}s)$$
$$= 50s$$

Variables

Variables	Units (usual)
Cycle	
Cycle count	cycles
Cycle time	Pico/nano seconds
Period	Pico/nano seconds
Rate	
Cycles per instruction	
Instruction count	
Execution Time	Seconds/minutes

If a program needs $5X10^9$ cycles, and the cycle time 100 $picoseconds$ on computer A.
Then the execution time ?

$$Execution\ time$$
$$= cycle\_count\ X\ cycle\_time$$
$$= (5X10^9)X\ (100\ X\ 10^{-12}s)$$
$$= 50s$$

Execution time

Variables	Units (usual)
Cycle	
Cycle count	cycles
Cycle time	Pico/nano seconds
Period	Pico/nano seconds
Rate	
Cycles per instruction	
Instruction count	
Execution Time	Seconds/minutes

$$Execution\ time = cycle\_count\ X\ cycle\_time$$

Execution time

Variables	Units (usual)
Cycle	
Cycle count	cycles
Cycle time	Pico/nano seconds
Period	Pico/nano seconds
Rate	
Cycles per instruction	
Instruction count	
Execution Time	Seconds/minutes

$$Execution\ time = cycle\_countX\ cycle\_time$$

Instead of cycle time, you can be given rate.

Rate = number of cycles in a second

Execution time

Variables	Units (usual)
Cycle	
Cycle count	cycles
Cycle time	Pico/nano seconds
Period	Pico/nano seconds
Rate	Hz/MHz/GHz
Cycles per instruction	
Instruction count	
Execution Time	Seconds/minutes

$$Execution\ time = cycle\_count\ X\ cycle\_time$$

Instead of cycle time, you can be given rate.

Rate = number of cycles in a second

$$Rate\ = \frac{1}{(cycle\_time)}$$

If rate (clock rate) is 4 GHz, what is the cycle time?

Execution time

Variables	Units (usual)
Cycle	
Cycle count	cycles
Cycle time	Pico/nano seconds
Period	Pico/nano seconds
Rate	Hz/MHz/GHz
Cycles per instruction	
Instruction count	
Execution Time	Seconds/minutes

$$Execution\ time = cycle\_count\ X\ cycle\_time$$

Instead of cycle time, you can be given rate.

$$Rate\ = \frac{1}{(cycle\_time)}$$

If rate (clock rate) is 4 GHz, what is the cycle time?

$$cycle\_time\ = \frac{1}{(rate)} = \frac{1}{4\ *10^9 Hz}$$

$$= \frac{1}{4\ *10^9} s = \frac{1000}{4X10^{12}} s = \frac{1000}{4} X10^{-12} s$$

$$= 250\ picoseconds$$

Now we have cycle time, we can compute execution time

Execution time

Variables	Units (usual)
Cycle	
Cycle count	cycles
Cycle time	Pico/nano seconds
Period	Pico/nano seconds
Rate	Hz/MHz/GHz
Cycles per instruction	cycles
Instruction count	instruction
Execution Time	Seconds/minutes

$$Execution\ time = cycle\_count\ X\ cycle\_time$$

Instead of cycle count, you can be given, instruction count and **cycles per instruction(CPI)**

CPI = cycle count for one instruction

Cycle count for 100 instructions= 100 CPI

$$cycle\_count\ =\ instruction\_count\ X\ CPI$$

Performance Review

Our favorite program runs in 10 seconds on computer A, which has a 2 GHz clock.

Performance Review

Our favorite program runs in 10 seconds on computer A, which has a 2 GHz clock.

How many cycles are needed to run the program?

Performance Review

Our favorite program runs in 10 seconds on computer A, which has a 2 GHz clock.

How many cycles are needed to run the program?

Performance Review

Our favorite program runs in 10 seconds on computer A, which has a 2 GHz clock.

Execution time

Rate

How many cycles are needed to run the program?

Performance Review

Our favorite program runs in 10 seconds on computer A, which has a 2 GHz clock.

Execution time

Rate

How many cycles are needed to run the program?

$$Execution\ time = cycle\_count\ X\ cycle\_time$$

$$\frac{Execution\ time}{cycle\_time} = cycle\_count$$

$$cycle\_time = \frac{1}{Rate} = \frac{1}{2X10^9}s = 500X10^{-12}s = 500ps$$

$$cycle\_count = \frac{10}{500X10^{-12}} = \frac{10000}{500X10^{-9}} = 20X10^9 = 20\ billion\ cycles$$

Our favorite program runs in 10 seconds on computer A, which has a 2 GHz clock. We are trying to help a computer designer build a computer, B, which will run this program in 6 seconds. The designer has determined that a substantial increase in the clock rate is possible, but this increase will affect the rest of the CPU design, causing computer B to require 1.2 times as many clock cycles as computer A for this program. What clock rate should we tell the designer to target?

$$cycle\_count = 20X10^9\ cycles$$

Our favorite program runs in 10 seconds on computer A, which has a 2 GHz clock. We are trying to help a computer designer build a computer, B, which will run this program in 6 seconds. The designer has determined that a substantial increase in the clock rate is possible, but this increase will affect the rest of the CPU design, causing computer B to require 1.2 times as many clock cycles as computer A for this program. What clock rate should we tell the designer to target?

$$cycle\_count = 20X10^9\ cycles$$

Design a computer B

$$Execution\ time = 6s$$

$$cycle\_count = 1.2X(20X10^9)cycles = 24X10^9 cycles$$

Our favorite program runs in 10 seconds on computer A, which has a 2 GHz clock. We are trying to help a computer designer build a computer, B, which will run this program in 6 seconds. The designer has determined that a substantial increase in the clock rate is possible, but this increase will affect the rest of the CPU design, causing computer B to require 1.2 times as many clock cycles as computer A for this program. What clock rate should we tell the designer to target?

$$cycle\_count = 20X10^9\ cycles$$

Design a computer B

$$Execution\ time = 6s$$

$$cycle_{count} = 1.2X(20X10^9)cycles = 24X10^9\ cycles$$

$$\boldsymbol{Rate} = ?$$

$$Execution\ time = cycle\_count\ X\ cycle\_time$$

$$cycle\_time = \frac{6}{24X10^9} = \frac{1}{4X10^9}\ s$$

Our favorite program runs in 10 seconds on computer A, which has a 2 GHz clock. We are trying to help a computer designer build a computer, B, which will run this program in 6 seconds. The designer has determined that a substantial increase in the clock rate is possible, but this increase will affect the rest of the CPU design, causing computer B to require 1.2 times as many clock cycles as computer A for this program. What clock rate should we tell the designer to target?

$$cycle\_count = 20X10^9\ cycles$$

Design a computer B

$$Execution\ time = 6s$$

$$cycle_{count} = 1.2X(20X10^9)cycles = 24X10^9\ cycles$$

$$\boldsymbol{Rate} = ?$$

$$Execution\ time = cycle\_count\ X\ cycle\_time$$

$$cycle\_time = \frac{6}{24X10^9} = \frac{1}{4X10^9}s$$

$$Rate = \frac{1}{cycle\_time} = 4X10^9Hz = 4GHz$$