Lecture 4
Chapter 4: The Instruction Set Architecture

CPS310

Computer Organization II

WINTER 2022

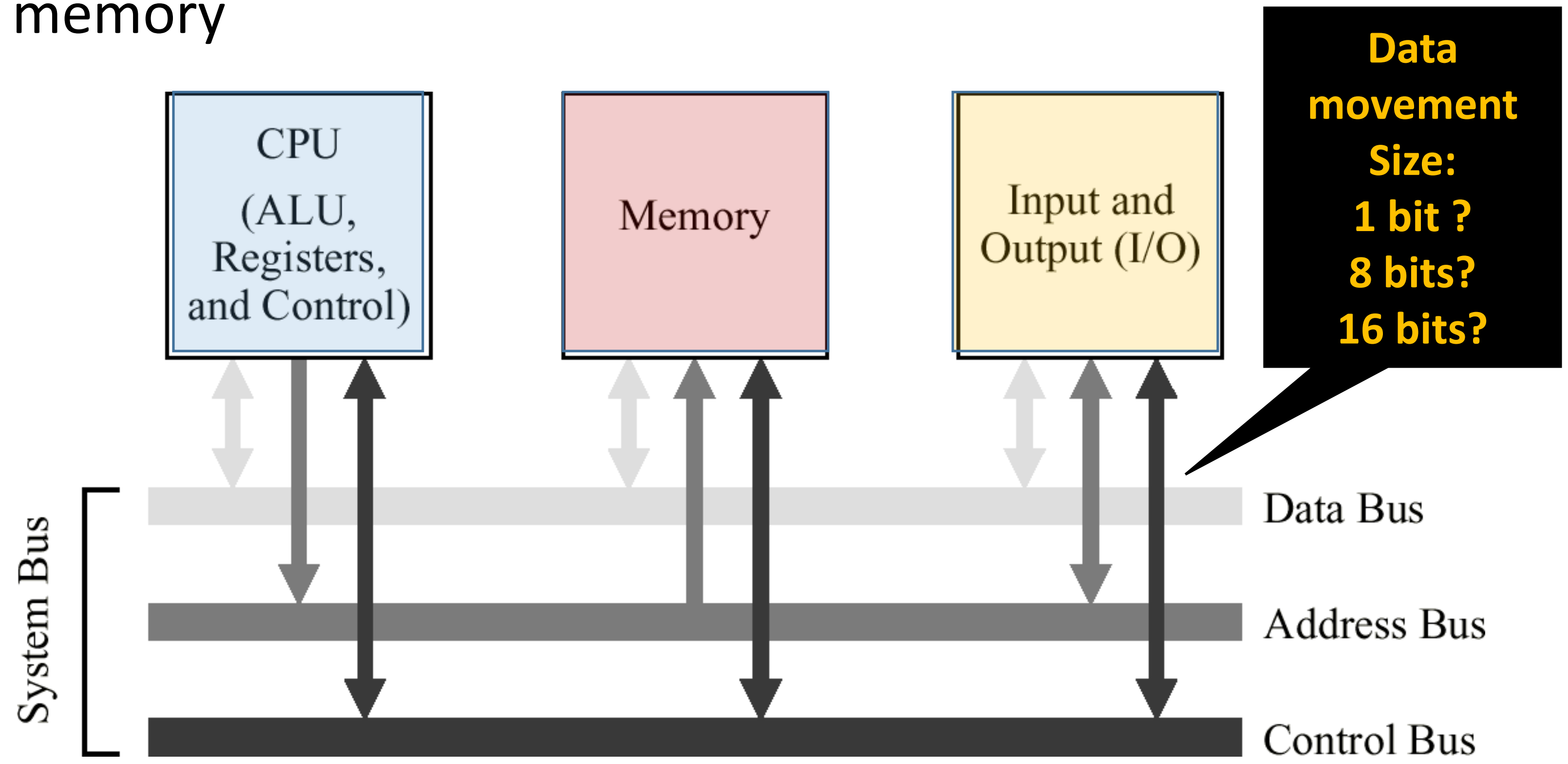
© Dr. A. Sadeghian

The Instruction Set Architecture

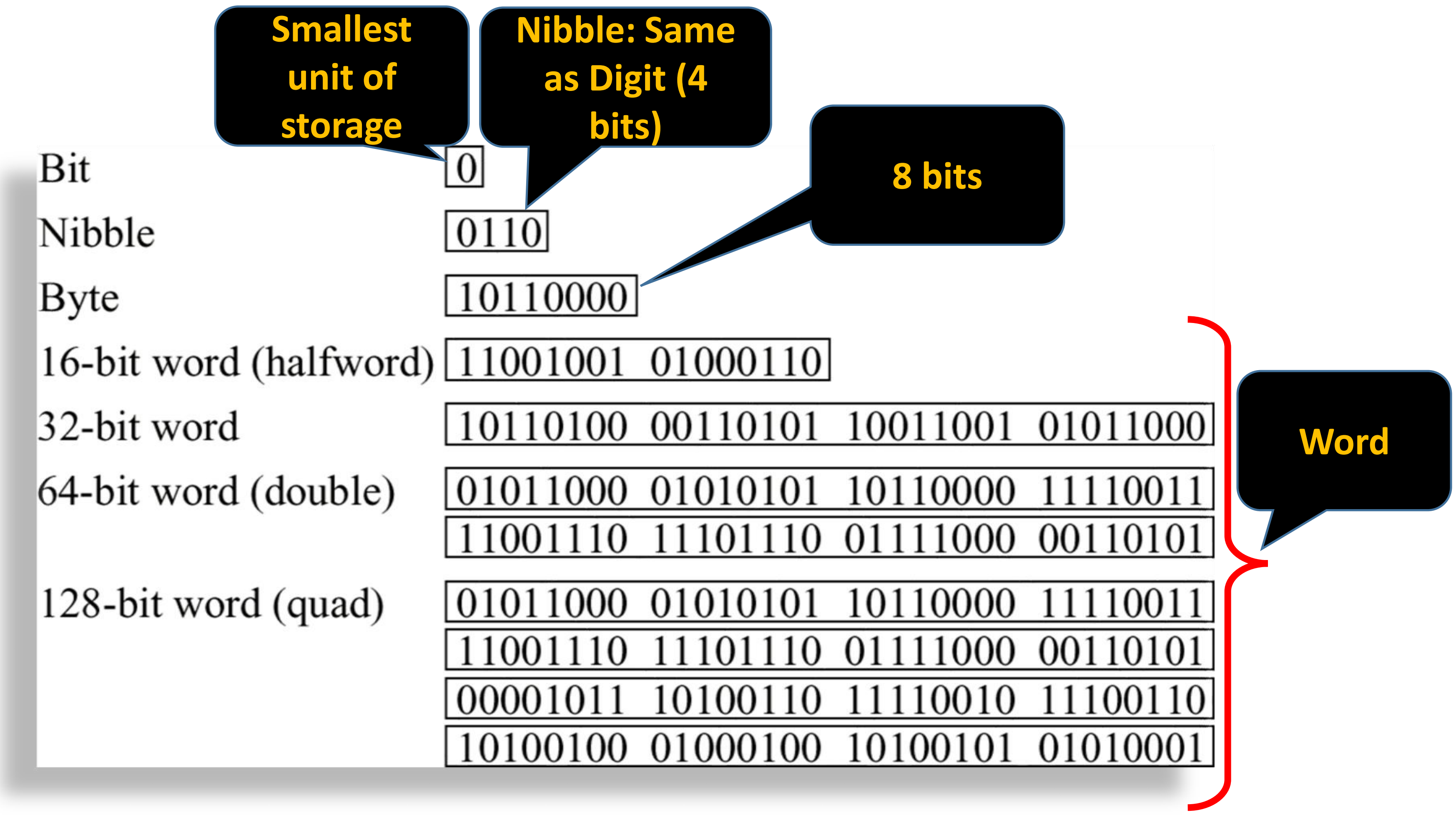
- The ***Instruction Set Architecture* (ISA)** view of a machine corresponds to the machine and assembly language levels.
- A ***compiler*** translates a high level language, which is architecture independent, into assembly language, which is architecture dependent.
- An ***assembler*** translates assembly language programs into executable binary codes.

The System Bus Model of a Computer System, Revisited

- A compiled program is copied from a hard disk to the memory
- The CPU reads instructions and data from the memory, executes the instructions, and stores the results back into the memory



Common Sizes for Data Types


Smallest unit of storage
Nibble: Same as Digit (4 bits)
8 bits
Word

Bit	0
Nibble	0110
Byte	10110000
16-bit word (halfword)	11001001 01000110
32-bit word	10110100 00110101 10011001 01011000
64-bit word (double)	01011000 01010101 10110000 11110011
	11001110 11101110 01111000 00110101
128-bit word (quad)	01011000 01010101 10110000 11110011
	11001110 11101110 01111000 00110101
	00001011 10100110 11110010 11100110
	10100100 01000100 10100101 01010001

Byte-Addressable Machine

- In a **byte-addressable machine**, the smallest data that can be accessed in memory is the byte

Example:

Using address 0x2048 we can access the memory content whose address is 0x2048

1 BYTE

Address	*Memory*
0x2048	0011 1000
0x2049	0101 1010
0x204A	0000 0011
0x204B	1111 0100

Byte-Addressable Machine

- **Single-byte data** is stored in a single memory location (eg, we need 1 byte to store 0011 1000)

- **Multi-byte data** is stored as a sequence of bytes in a number of memory locations
  (eg, we need 4 bytes to store
  0011 1000
  1010 1010
  0000 0011
  1111 0100)

1 BYTE

Address	*Memory*
0x2048	**0011 1000**
0x2049	**0101 1010**
0x204A	**0000 0011**
0x204B	**1111 0100**

Single-Byte data and Memory

- **How to access the data** (what is the address)?

  The address of the data is the address of the memory location holding the single byte data

- **How to store/read the data?**

  There is only one way to store/read the data

Address	Memory Content
0x2048	1010 1010

Multi-Byte data and Memory

- How to access the data (what is the address)?

  The address is the same as the address of the byte with the lowest address

- How to store/read the data?

  Big-Endian vs Little-Endian

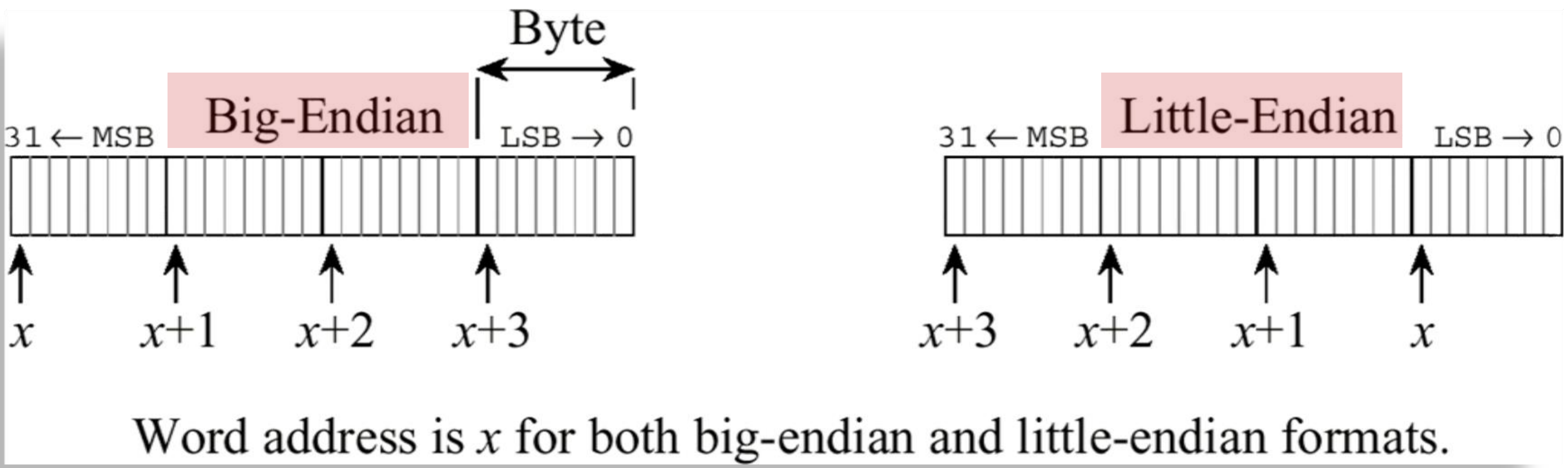
Address	Memory Content
0x2048	0000 1111
0x2049	0101 0101
0x204A	1010 1010
0x204B	1111 0000

Big-Endian and Little-Endian Formats

How to store multi-byte in memory - 2 choices:

***big-endian**:* most significant byte is stored first

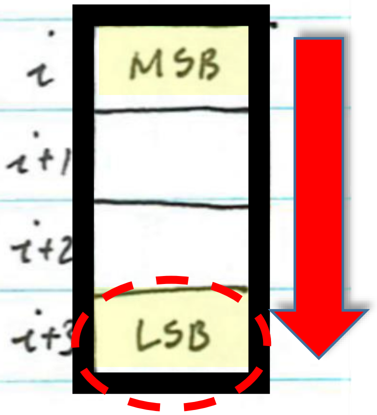
little-endian: least significant byte is stored first



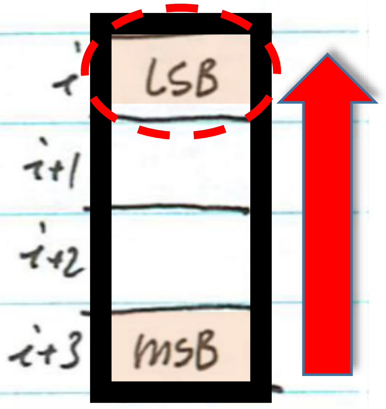
Big-Endian vs Little-Endian

HOW TO STORE INFO IN MEMORY?

BIG-ENDIAN: MOST SIGNIFICANT BYTE AT LOWEST ADDRESS



LITTLE-ENDIAN: LEAST SIGNIFICANT BYTE AT LOWEST ADDRESS



Big-Endian vs Little-Endian - Example

Address	Memory
0x2048	0 4
0x2049	0 3
0x204A	0 2
0x204B	0 1

0000 0100

Big Endian

MSB

LSB

How to store
0x 04 03 02 01
in memory
location 0x 2048 ?

Address	Memory
0x2048	0 1
0x2049	0 2
0x204A	03
0x204B	04

Little Endian

ARC – A RISC Computer

The ARC ISA is a subset of the SPARC ISA

SPARC - **S**calable **P**rocessor **Arc**hitecture
a **R**educed **I**nstruction **S**et **C**omputing (RISC) ISA

RISC vs CISC

RISC: <u>R</u>educed <u>I</u>nstruction <u>S</u>et <u>C</u>omputer

- Less instructions
- Simpler instructions (typically perform one task)
- More registers
- Easy to pipeline
- Work with registers more

RISC vs CISC

CISC: <u>C</u>omplex <u>I</u>nstruction <u>S</u>et <u>C</u>omputer

- More instructions
- Complex instructions

  (can perform more than one task)
- Less registers
- Not easy to pipeline
- Work with memory more

SPARC

- An instruction set architecture (ISA)
- With 32-bit integer and 32, 64, 128-bit IEEE Standard 754 floating-point as its principal data types

- It defines general-purpose integer, floating-point, and special state/status registers
- 72 basic instruction operations, all encoded in 32-bit wide instruction formats

SPARC Processor

- Typically comprises an integer unit (**IU**), a floating-point unit (**FPU**), an optional coprocessor (**CP**), each with its own registers

- **IU**: (aka ALU) is the central part of the processor. In its most basic design it is a combinational logic circuit performing arithmetic and logical operations on **integer** numbers

- **FPU**: A (**aka** a math coprocessor) designed to perform operations on **floating point** numbers

SPARC Co-Processor (CP)

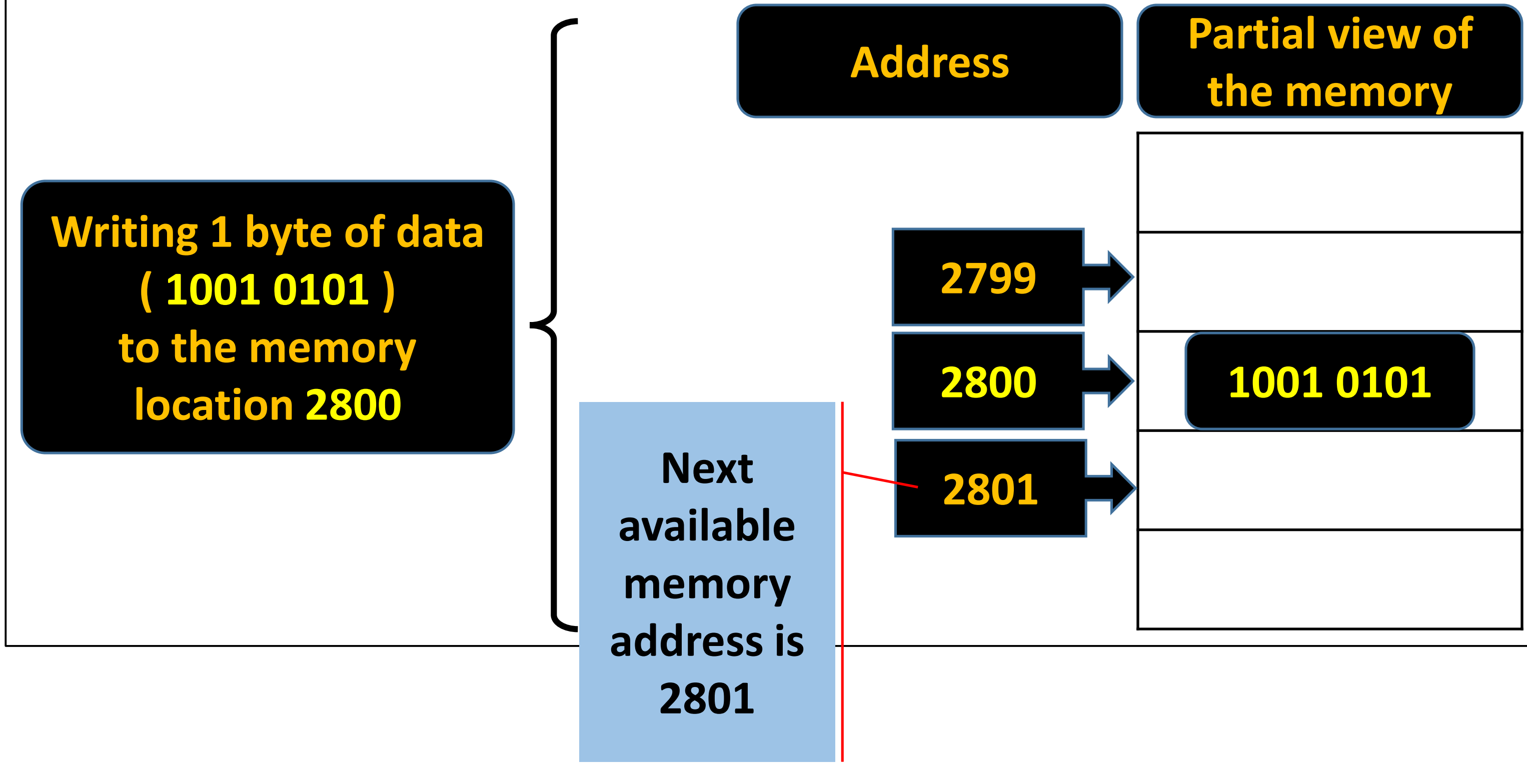
- A processor to add to the capabilities of the main microprocessor
- Maximum concurrency between integer, floating-point, and coprocessor instruction execution
- All of the registers — with the possible exception of the coprocessor’s — are 32 bits wide
- Instruction operands are generally single registers, register pairs, or register quadruples

Side note – single-byte vs multi-byte movement

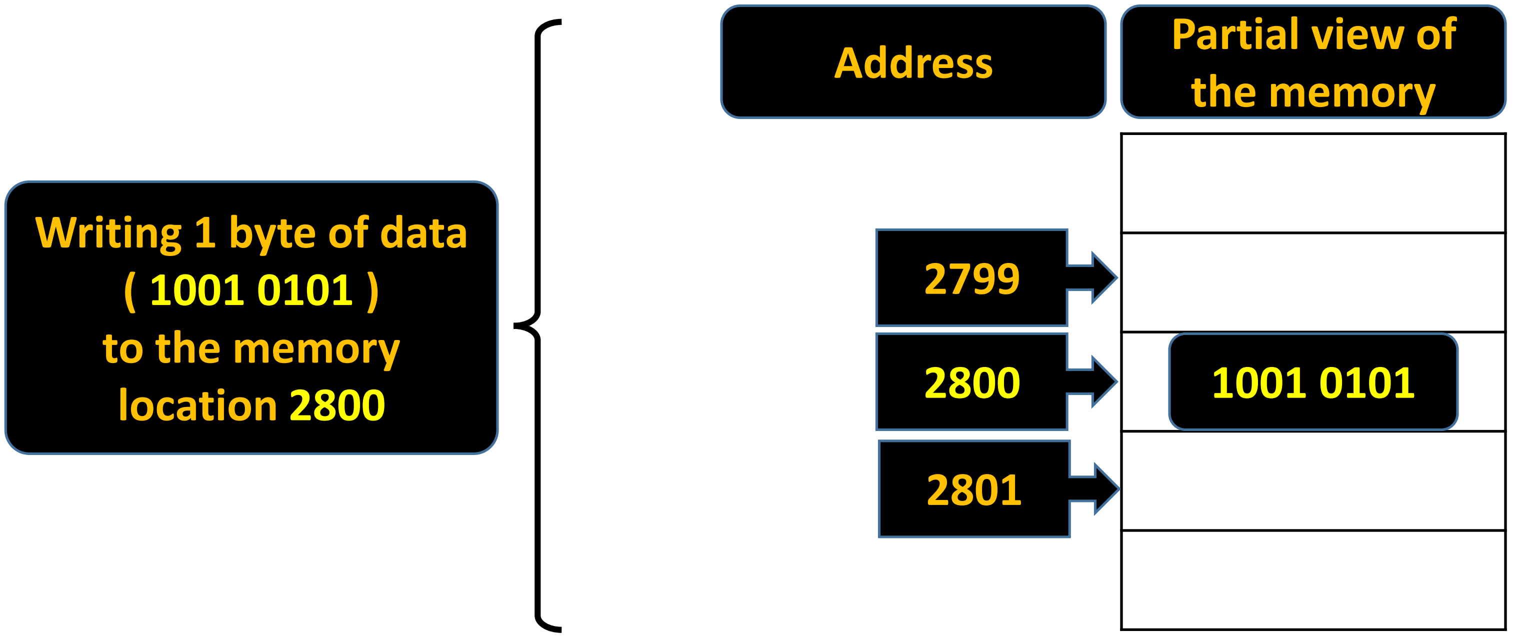
Some CPUs work with single byte of data

They either:

- Store 1 byte of data to the memory, or
- Read 1 byte of data from the memory


Address
Partial view of the memory
Writing 1 byte of data (1001 0101) to the memory location 2800
2799
2800
2801
1001 0101
Next available memory address is 2801

Side note – single-byte vs multi-byte movement



All memory addresses are accessible

All memory addresses can be used for reading and writing

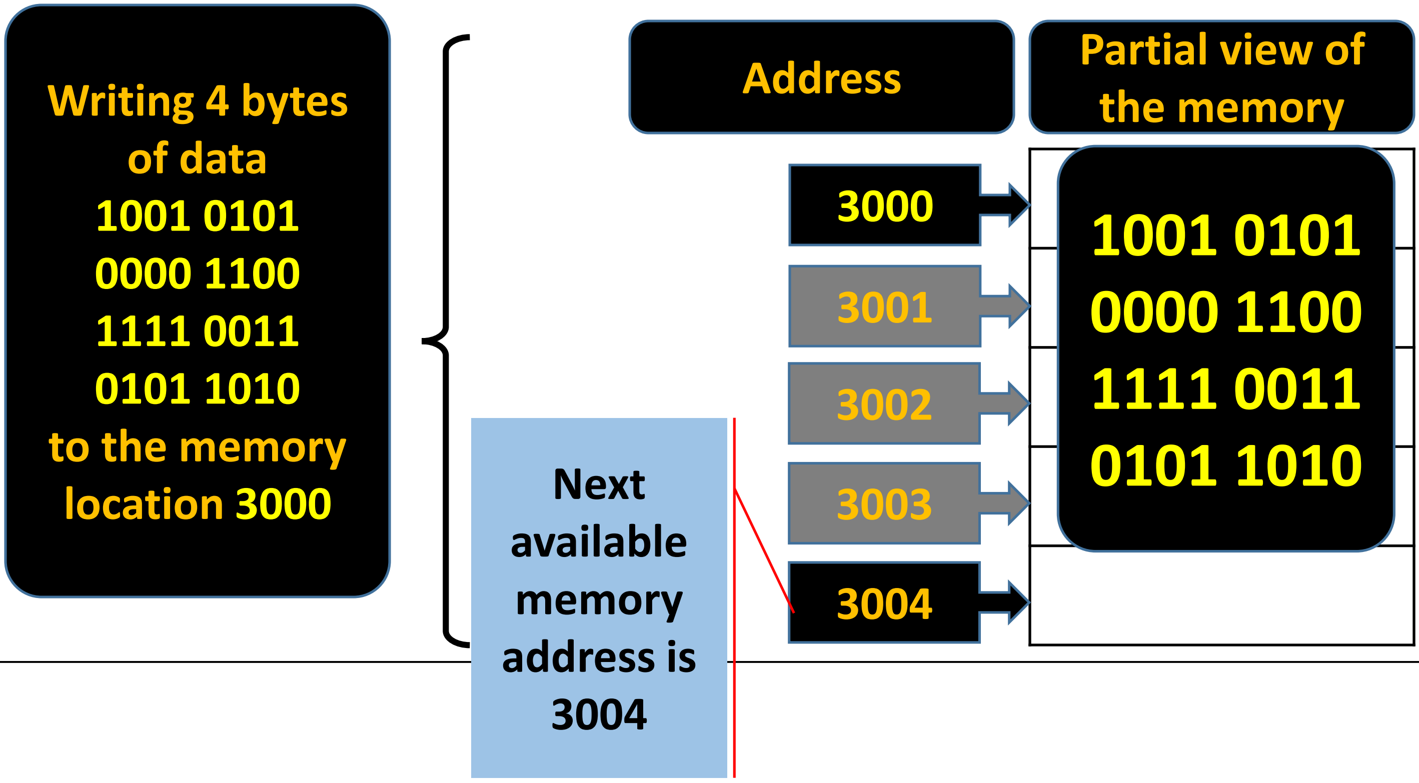
For example,
2799
2800
2801
…

Side note – single-byte vs multi-byte movement

Some CPUs work with multi-byte of data

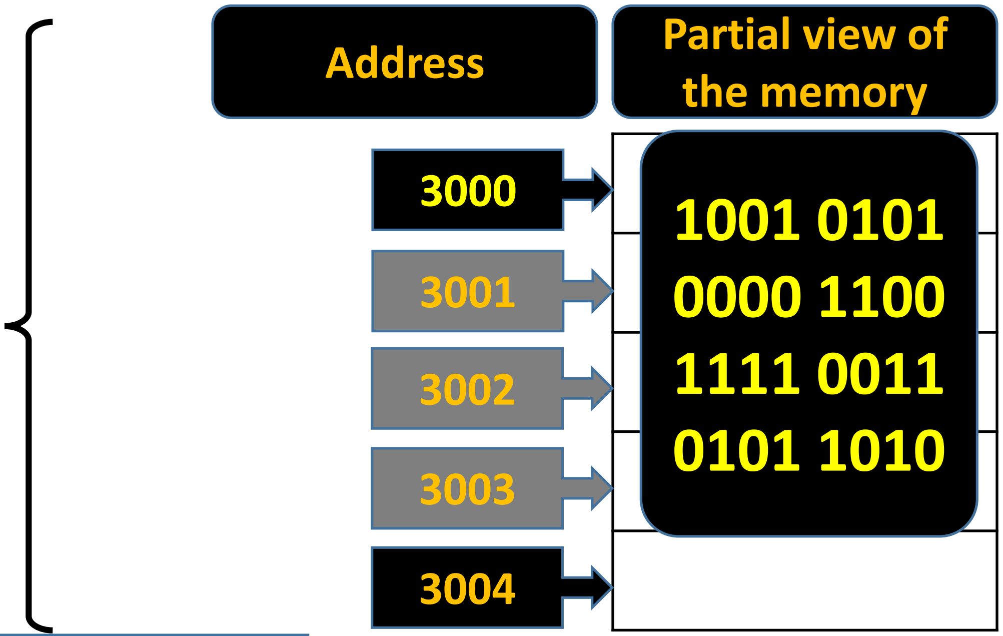
They either:

- Store multi-byte of data to the memory, or
- Read multi-byte of data from the memory



Side note – single-byte vs multi-byte movement

Writing 4 bytes of data
1001 0101
0000 1100
1111 0011
0101 1010
to the memory location 3000

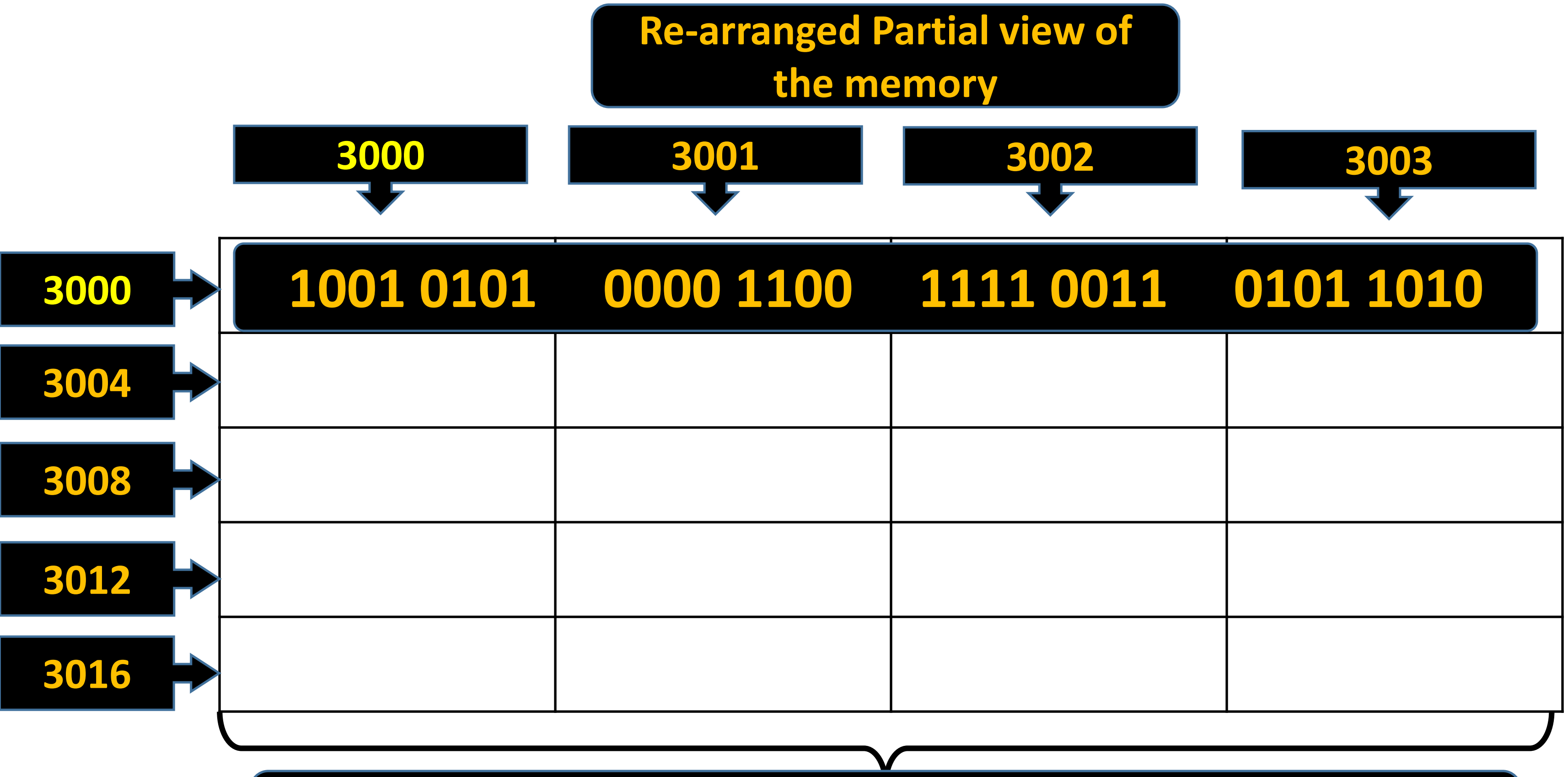


All memory addresses are accessible

Only memory addresses that are multiple of 4 should be used for reading and writing

For example,
3000
3004
3008
...

Side note – single-byte vs multi-byte movement



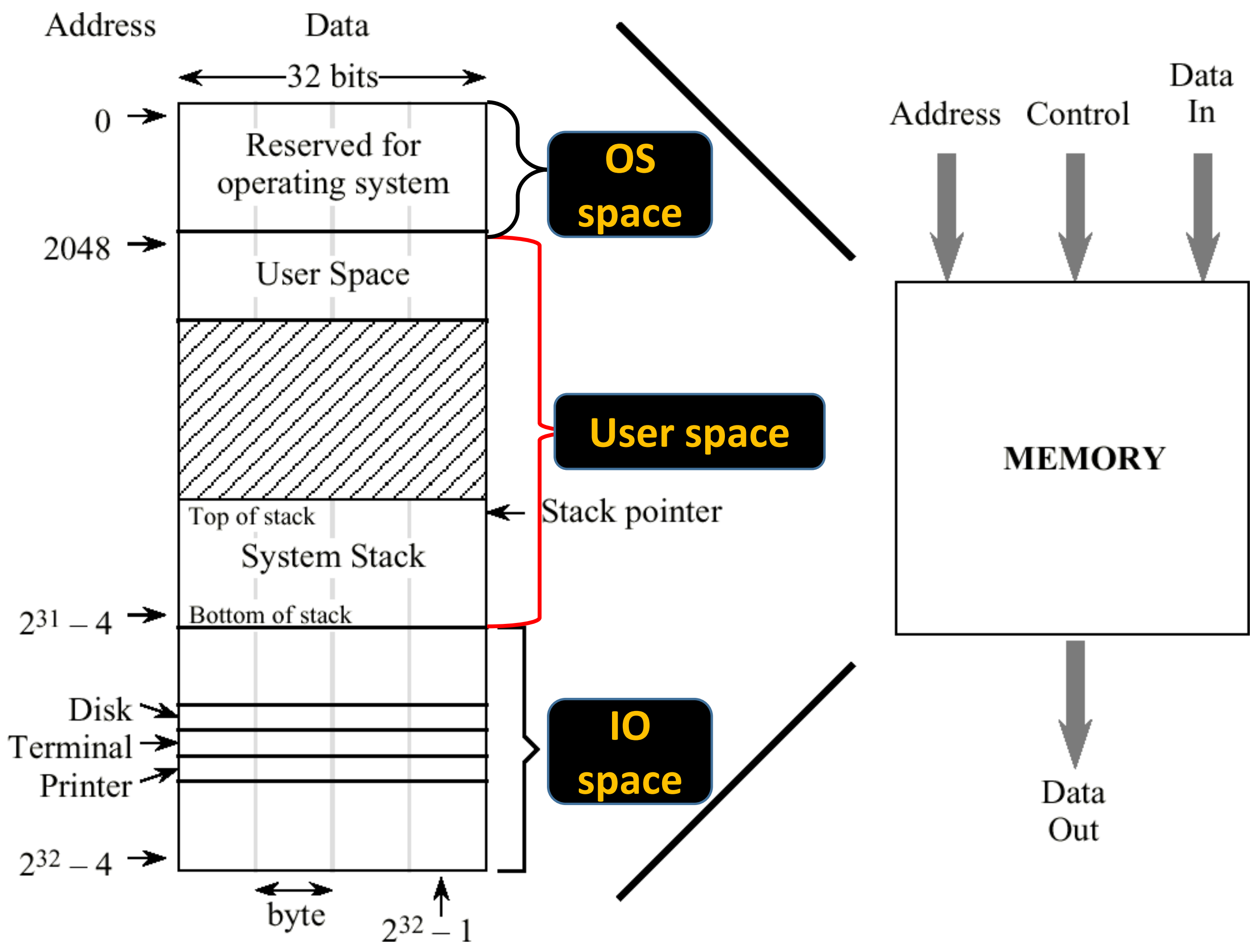
Writing 4 bytes to the memory location 3000

This is known as 4-byte addressable memory
i.e., at any given time, CPU reads 4 bytes from the memory or write 4 byte to the memory

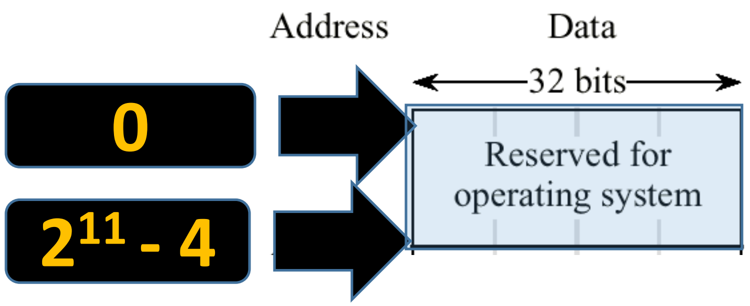
ARC

- Addresses are **32** bits (0 to 2^{32}-1)
- **32**-bit byte-addressable memory
- can manipulate **32**-bit data
- the address of a **32**-bit word is the address of its byte with the lowest address
- Big-Endian (MSB in the lowest address)

Memory Map for the ARC



ARC Memory Map – Reserved Locations, OS Space



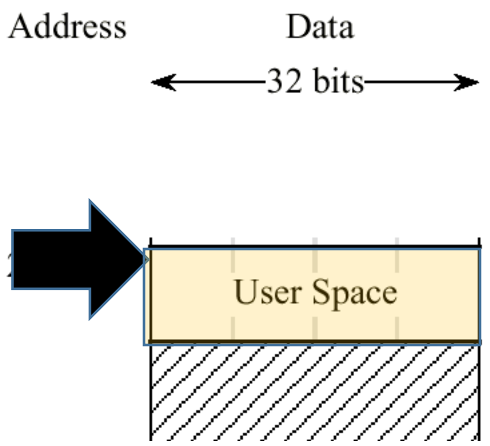
Reserved Locations
OS Space

The top portion of the memory from address 0 to 2044 is reserved for the operating system

Not to be used by the user

ARC Memory Map – User Space

$2^{11} = 2048$



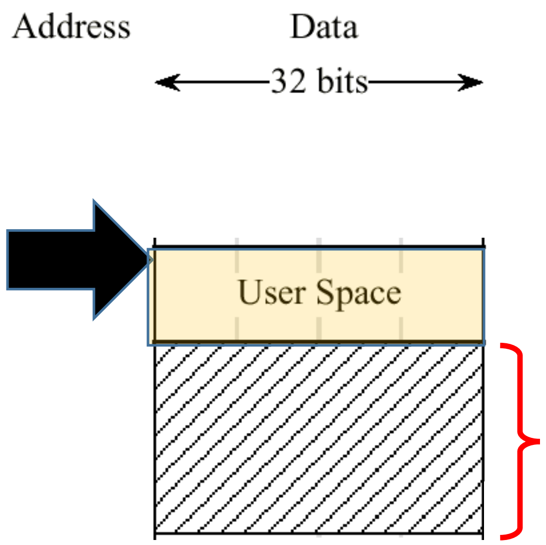
User Space

To be used by the user
Code & Data & Stack

Starting from address 2048

ARC Memory Map – User Space

$2^{11} = 2048$



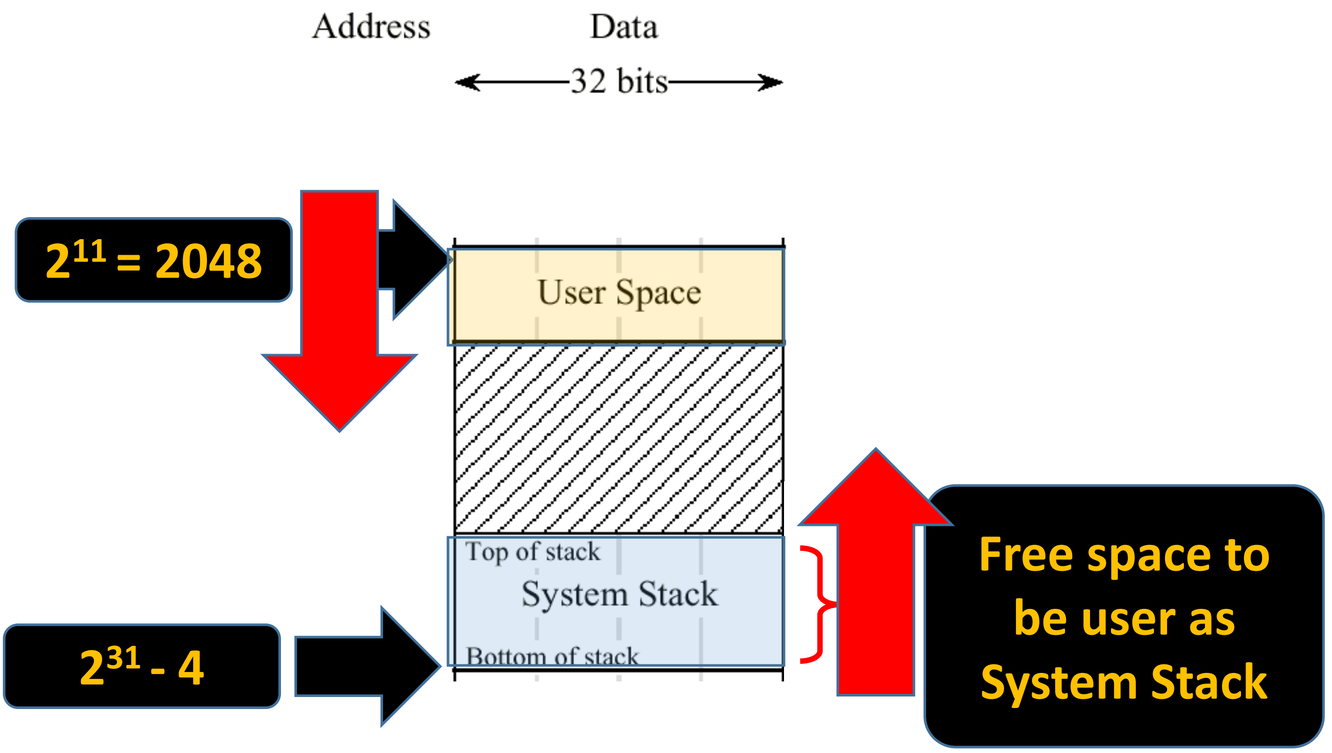
Free space that can further be used by the user

User Space

To be used by the user Code & Data & Stack

Starting from address 2048

ARC Memory Map – Stack Space



$2^{11} = 2048$

$2^{31} - 4$

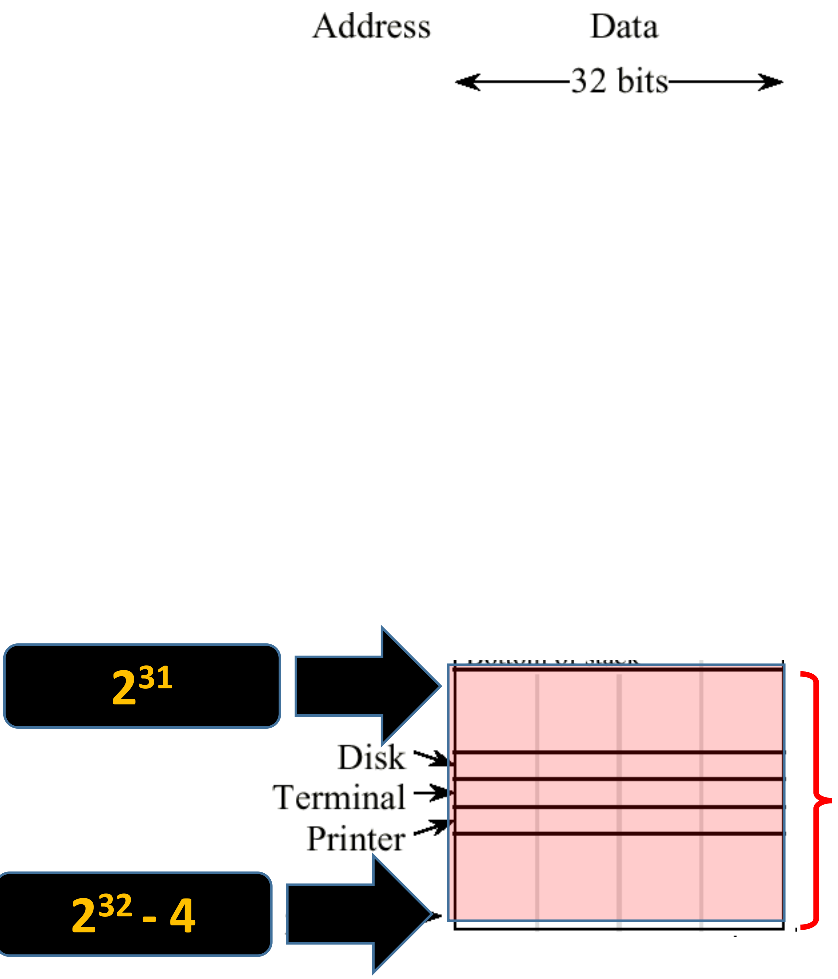
Free space to be user as System Stack

User Space + Stack

To be used by the user Code & Data & Stack

Bottom of the user space (bottom of the Stack): from address $2^{31} - 4$

ARC Memory Map – I/O Space



Free space to be used as I/O

I/O Space

To be used by the user Code & Data & Stack

Bottom of the user space (bottom of the Stack): from address $2^{31} - 4$

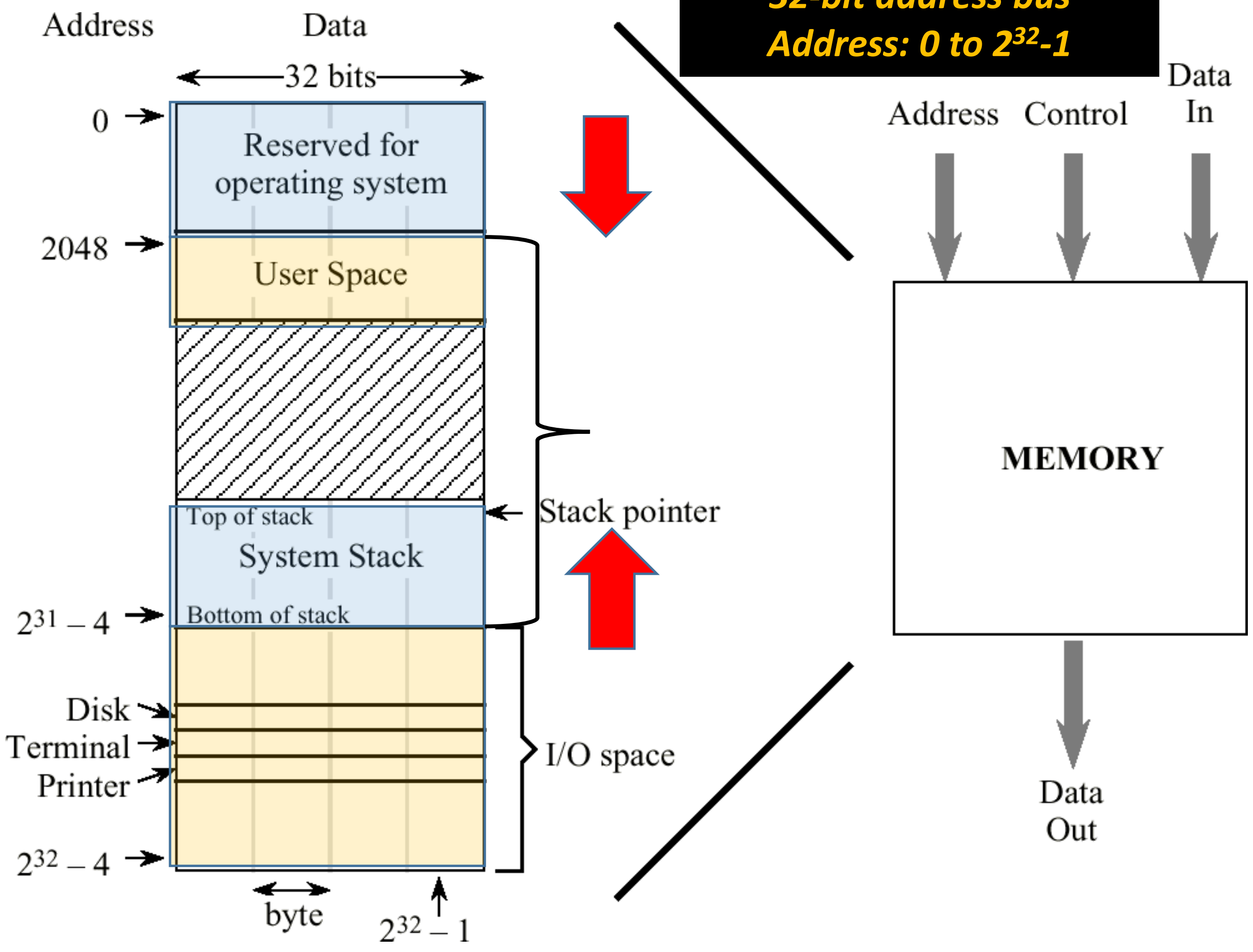
Memory Map for the ARC

Memory locations are arranged linearly in consecutive order

Each numbered locations corresponds to an ARC word

The unique number that identifies each word is referred to as its address.

32-bit address bus
Address: 0 to 2^{32}-1



Memory Mapped I/O

- I/O devices are treated as memory locations

- Have unique addresses in the memory

- Memory Read/Write commands can be used for
 Reading to
 and
 Writing from
 the devices

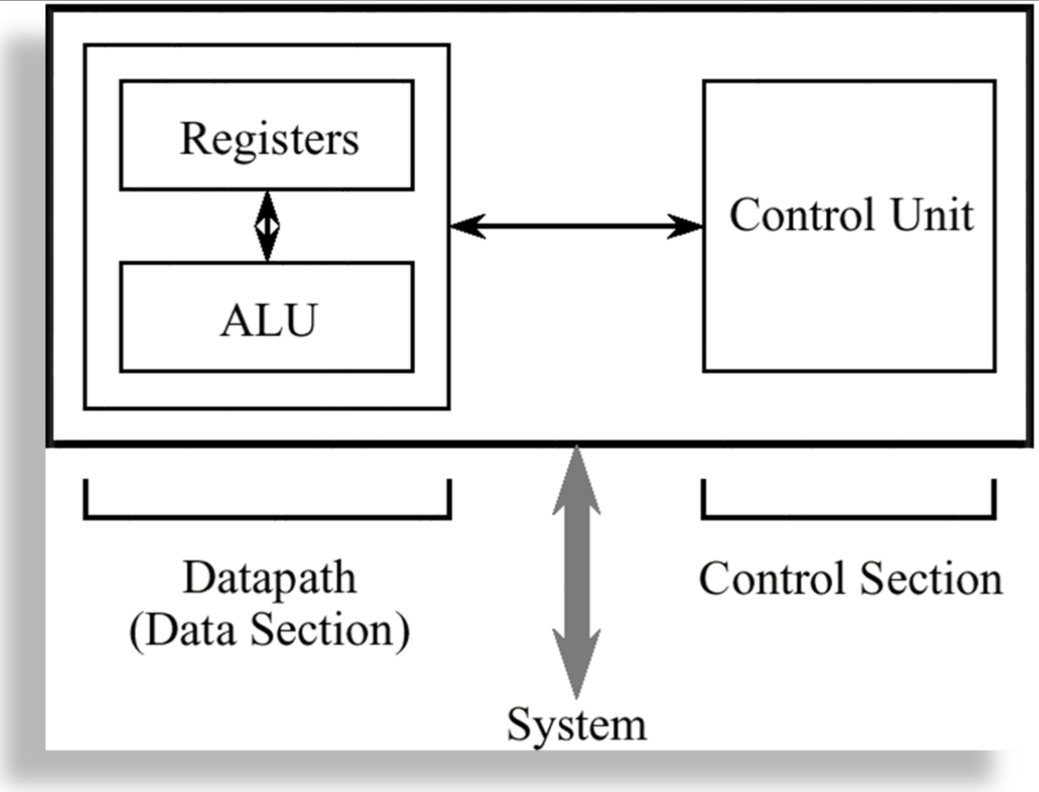
Abstract View of a CPU

data section (AKA datapath):

- ***registers and an ALU***

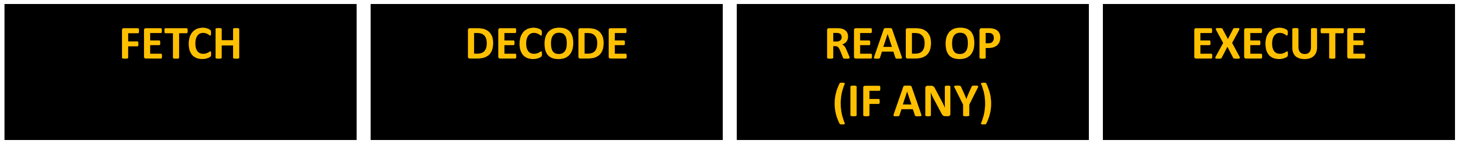
control section:

- **interpret instructions**
- **effect register transfers**
- **execute the instructions**



The Fetch-Execute Cycle

The steps that the **control unit** carries out in executing a program are:



(1) Fetch the next instr to be executed from memory
(2) Decode the opcode
(3) Read operand(s) from main memory, if any
(4) Execute the instruction and store results
(5) Go to step 1

Non - Pipelining

	Cycle 1	Cycle 2	Cycle 3	Cycle 4
1st Instruction	FETCH	DECODE	READ OP (IF ANY)	EXECUTE
	Cycle 5	Cycle 6	Cycle 7	Cycle 8
2nd Instruction	FETCH	DECODE	READ OP (IF ANY)	EXECUTE

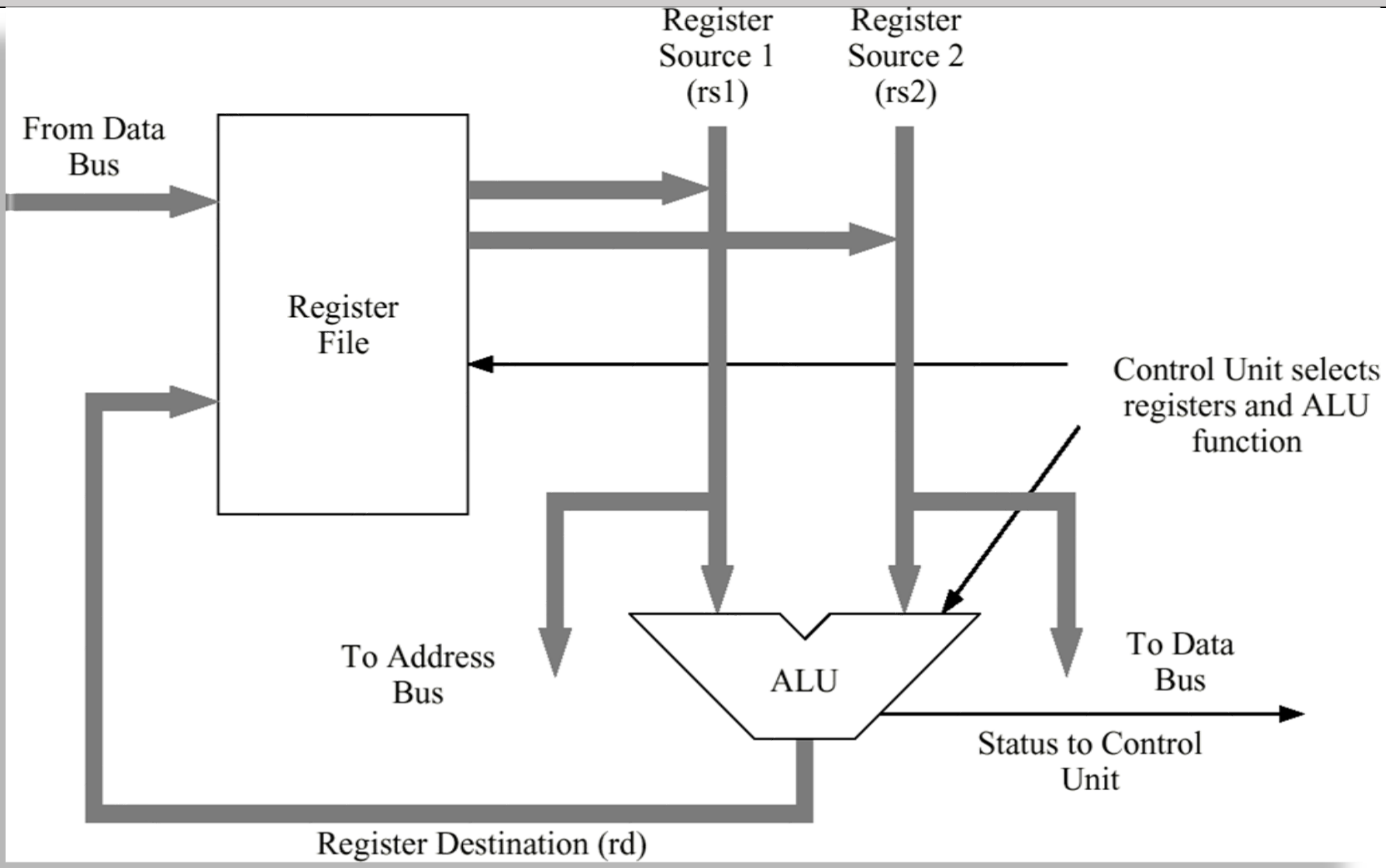
Pipelining

	Cycle 1	Cycle 2	Cycle 3	Cycle 4	Cycle 5	Cycle 6
1st Instruction	FETCH	DECODE	READ OP (IF ANY)	EXECUTE		
2nd Instruction		FETCH	DECODE	READ OP (IF ANY)	EXECUTE	
3rd Instruction			FETCH	DECODE	READ OP (IF ANY)	EXECUTE

Register File

- A small fast memory
- Very similar to the main memory
- But separate from the main memory
- Located inside the CPU
- Hence faster
- From few registers to few thousands registers
- Each register has a distinct designation
- In ARC, 5-bit designation/address, for 32 registers

An Example Datapath



The ARC datapath is made up of a collection of registers known as the register file and the arithmetic and logic unit (ALU)

Instruction Set

- A collection of instructions that a processor can execute
- Difference processors have different instructions
- They typically differ in
  - Size of the instruction
  - Type of operations that they perform
  - Complexity of the instructions
  - Type of operands they use