Topic V02

Hexadecimal Notation and
Storing Data in Memory

Hexadecimal

Value	Binary	Digit
0	0000	
1	0001	
2	0010	
3	0011	
4	0100	
5	0101	
6	0110	
7	0111	

Value	Binary	Digit
8	1000	
9	1001	
10	1010	
11	1011	
12	1100	
13	1101	
14	1110	
15	1111	

Example: ECA8 6420

1110 1100 1010 1000 0110 0100 0010 0000

Hexadecimal Example

How do you represent the number $+19_{10}$ in 32-bits?

19 = 16 + 2 + 1

19 = 0000 0000 0000 0000 0000 0000 0001 0011

19 = 0x00000013

How do you represent the number $+19_{10}$ in hexadecimal?

Hexadecimal Example

How do you represent the number -105_{10} in 32-bits?

105 = 64 + 32 + 8 + 1

105 = 0000 0000 0000 0000 0000 0000 0110 1001

$\overline{105}$ = 1111 1111 1111 1111 1111 1111 1001 0110

\+ 1

-105 = 1111 1111 1111 1111 1111 1111 1001 0111

-105= 0xFFFFFF97

How do you represent the number -105_{10} in hexadecimal?

Hexadecimal Example

How do you represent the number -105_{10} in 32-bits?

105 = 64 + 32 + 8 + 1

105 = 0000 0000 0000 0000 0000 0000 0110 1001

$\overline{105}$ = 1111 1111 1111 1111 1111 1111 1001 0110

\+ 1

-105 = 1111 1111 1111 1111 1111 1111 1001 0111

Big End

-105= 0xFFFFFF97

Little End

To store this number in memory we group the bits into 8-bit groups called <u>bytes</u>

Memory

The memory of a computer is simply an array of bytes.

Thus, there is an index associated with each byte stored in memory.

The index of a given byte in memory is called its **address**.

The idea of an address

Address of a byte in memory

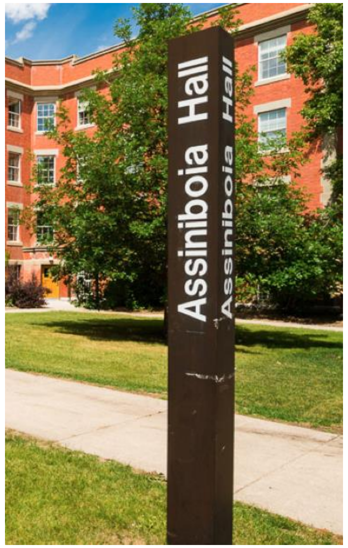
0x30004000 0x02

0x00000000 0x42

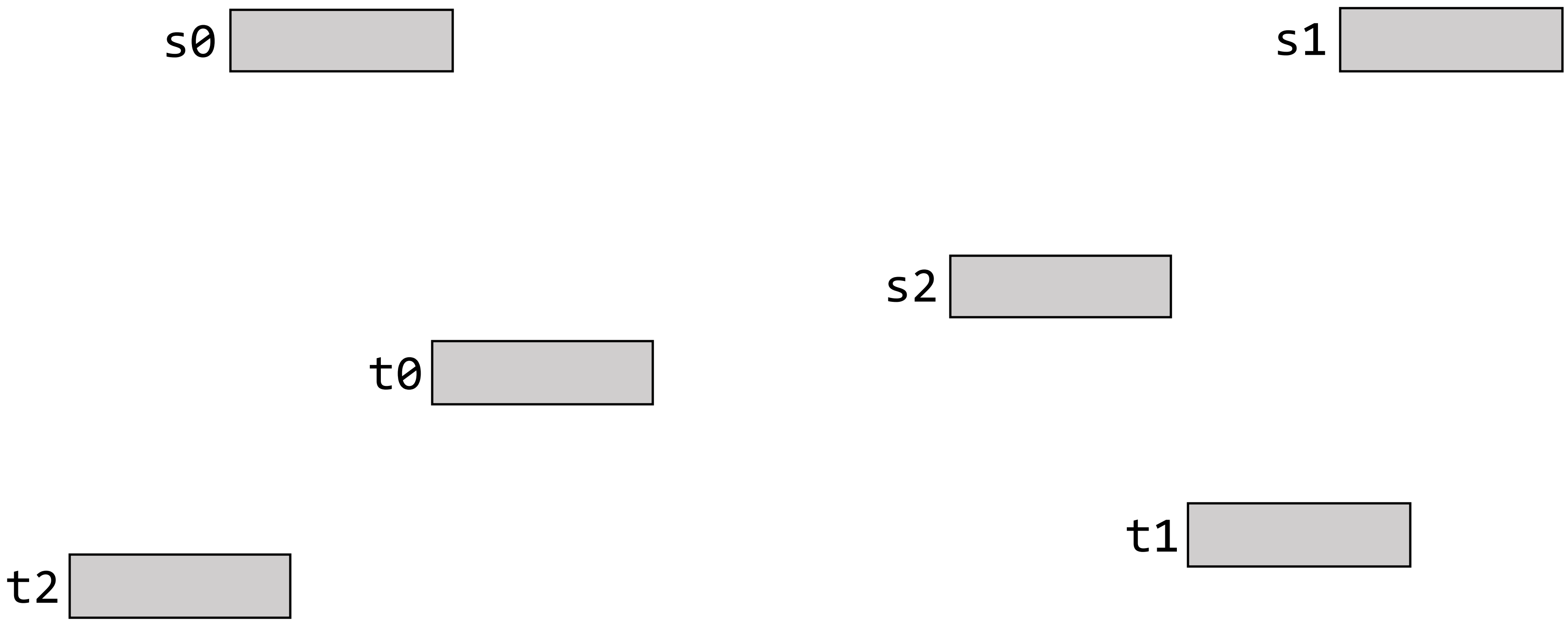
0xFE007C00 0x11

0x00000001 0xA7

Some places do not have an address, they have a name:



A register in a processor does not have an address, it has a name



A byte in memory

A byte stored in memory has a value and an address.

In a 32-bit machine the address has 32 bits, and thus it is represented using eight hexadecimal digits.

For example, if the memory address `0x30004000` contains the byte `0100 0010`, on paper we represent this as:

Address	Value
`0x30004000`	`0x42`

In this drawing each box stores 1 byte.

Memory Organization

In RISC-V an integer is represented by 4 bytes

A memory address references a single byte

The difference between the addresses
of two consecutive integers in memory is 4

Store the numbers $+19_{10}$ and -105_{10} in memory,
in consecutive addresses,
starting at address `0x10001000`:
($+19_{10} = 00010011_2 =$ `0x00000013`)
($-105_{10} = 10010111_2 =$ `0xFFFFFF97`)

This value is represented by four bytes:
`0x00, 0x00, 0x00, 0x13`

	Address	Value
↑ High Address	0x10001007	0x97
	0x10001006	0xFF
	0x10001005	0xFF
	0x10001004	0xFF
	0x10001003	0x13
	0x10001002	0x00
	0x10001001	0x00
↓ Low Address	0x10001000	0x00

Each box stores a single byte.

In this example:

The memory address of $+19_{10}$ is `0x10001000`

The memory address of -105_{10} is `0x10001004`

Endianess

In the previous example, there are two ways to store the numbers:

$+19_{10}$ = 0x0000 0013 at the address `0x10001000`

-105_{10} = 0xFFFF FF97 at the address `0x10001004`

Address	Value
0x10001007	0x97
0x10001006	0xFF
0x10001005	0xFF
0x10001004	0xFF
0x10001003	0x13
0x10001002	0x00
0x10001001	0x00
0x10001000	0x00

Address	Value
0x10001007	0xFF
0x10001006	0xFF
0x10001005	0xFF
0x10001004	0x97
0x10001003	0x00
0x10001002	0x00
0x10001001	0x00
0x10001000	0x13

What is the difference?

Little-End and Big-End

$+19_{10}$ = 0000 0000 0000 0000 0000 0000 0001 0011 (binary)

Big end of $+19_{10}$

Little end of $+19_{10}$

$+19_{10}$ = 0x0000 0013 (hexadecimal)

-105_{10} = 1111 1111 1111 1111 1111 1111 1001 0111 (binary)

Big end of -105_{10}

Little end of -105_{10}

$+19_{10}$ = 0xFFFF FF97 (hexadecimal)

Endianess

The question is: which **end** of the integer do we store at the lower address in memory?

Address	Value
0x10001007	0x97
0x10001006	0xFF
0x10001005	0xFF
0x10001004	0xFF
0x10001003	0x13
0x10001002	0x00
0x10001001	0x00
0x10001000	0x00

Big end of $+19_{10}$

Big-Endian Byte Order

Address	Value
0x10001007	0xFF
0x10001006	0xFF
0x10001005	0xFF
0x10001004	0x97
0x10001003	0x00
0x10001002	0x00
0x10001001	0x00
0x10001000	0x13

Little end of $+19_{10}$

Little-Endian Byte Order

RISC-V, DECstations and Intel 80x86 are little-endians
MIPS, Sun SPARC and IBM POWER are big-endians

RISC-V is Little Endian

Address	Value	Address	Value
0x1000100F	0x00	0x1000101F	0x00
0x1000100E	0x00	0x1000101E	0x00
0x1000100D	0x00	0x1000101D	0x00
0x1000100C	0x07	**0x1000101C**	0x0F
0x1000100B	0x00	0x1000101B	0x00
0x1000100A	0x00	0x1000101A	0x00
0x10001009	0x00	0x10001019	0x00
0x10001008	0x05	**0x10001018**	0x0D
0x10001007	0x00	0x10001017	0x00
0x10001006	0x00	0x10001016	0x00
0x10001005	0x00	0x10001015	0x00
0x10001004	0x03	**0x10001014**	0x0B
0x10001003	0x00	0x10001013	0x00
0x10001002	0x00	0x10001012	0x00
0x10001001	0x00	0x10001011	0x00
0x10001000	0x01	**0x10001010**	0x09

Least-significant byte at lowest address of a word

This is the word:
`0x0000 000B`

In a big-endian machine this would be the word:
`0x0B00 0000`

What we Have learned

Value	Binary	Digit	Value	Binary	Digit
0	0000	0	8	1000	8
1	0001	1	9	1001	9
2	0010	2	10	1010	a
3	0011	3	11	1011	b
4	0100	4	12	1100	c
5	0101	5	13	1101	d
6	0110	6	14	1110	e
7	0111	7	15	1111	f

	Address	Value
↑ High Address	0x10001007	0x97
	0x10001006	0xFF
	0x10001005	0xFF
	0x10001004	0xFF
	0x10001003	0x13
	0x10001002	0x00
	0x10001001	0x00
↓ Low Address	0x10001000	0x00

Address	Value
0x10001007	0x97
0x10001006	0xFF
0x10001005	0xFF
0x10001004	0xFF
0x10001003	0x13
0x10001002	0x00
0x10001001	0x00
0x10001000	0x00

Big end of $+19_{10}$

Big-Endian Byte Order

Address	Value
0x10001007	0xFF
0x10001006	0xFF
0x10001005	0xFF
0x10001004	0x97
0x10001003	0x00
0x10001002	0x00
0x10001001	0x00
0x10001000	0x13

Little end of $+19_{10}$

Little-Endian Byte Order