

# Topic V07

Large Constants  
(begin of Section 2.6)

# AND Operations (example)

You have:

s0 

0	0	1	0	1	1	0	0	1	0	0	1	1	1	0	1	0	0	1	1	1	0	1	1	1	0	0	0	0	1	0
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

You want:

s1 

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	1	1	0	1
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

First AND s0 bitwise with a mask to make all unwanted bits equal zero:

s0 

0	0	1	0	1	1	0	0	1	0	0	1	1	1	0	1	0	0	1	1	1	0	1	1	1	0	0	0	0	1	0
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

t0 

0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

t1 

0	0	0	0	0	0	0	0	1	0	0	1	1	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Now you shift the result 16 bits to the right:

s1 

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	1	1	0	1
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

# Handling Large Constants

To create the following mask:

t0

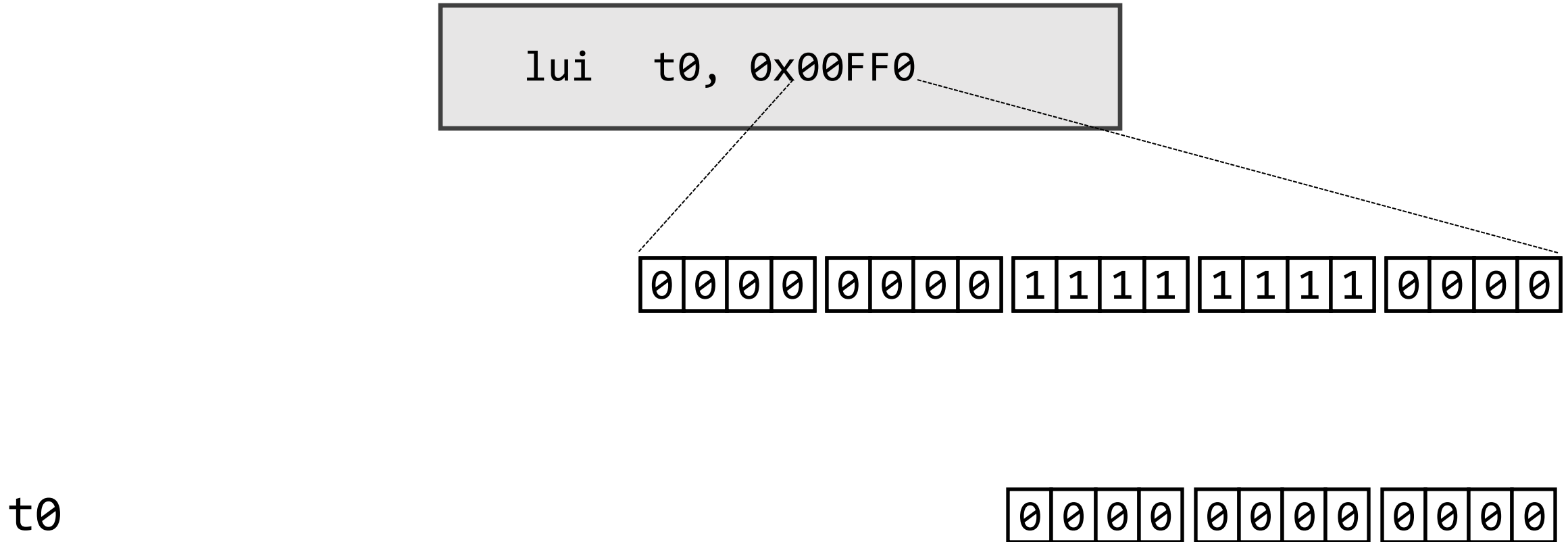
0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

We need to load the constant 0x00FF 0000 into t0

This can be done with a lui instruction:

```
lui    t0, 0x00FF0
```

How does the “load upper immediate” instruction work?



# Another Solution

To create the following mask:

t0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
----	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

We need to load the constant 0x00FF 0000 into t0

Another solution is to do the following:

## Load the constant 0x0FF into t0

Then shift  $t_0$  to the left by 16 bits

```
li    t0, 0xFF
sllli t0, t0, 16
```

t0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
----	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

You have:

s0	0	0	1	0	1	1	0	0	1	0	0	1	1	0	1	1	0	1	1	0	0	1	1	1	0	0	1	0
t0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
t1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
s1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

```
li    t0, 0xFF
slli  t0, t0, 16
and    t1, s0, t0
srli   s1, t1, 16
```

Another solution using only shifts

# AND Operations (example) Another Solution

You have:

s0 

0	0	1	0
---	---	---	---

1	1	0	0
---	---	---	---

1	0	0	1
---	---	---	---

1	1	0	1
---	---	---	---

0	0	1	1
---	---	---	---

1	1	0	1
---	---	---	---

1	1	0	0
---	---	---	---

0	0	1	0
---	---	---	---

You want:

s1 

0	0	0	0
---	---	---	---

0	0	0	0
---	---	---	---

0	0	0	0
---	---	---	---

0	0	0	0
---	---	---	---

0	0	0	0
---	---	---	---

0	0	0	0
---	---	---	---

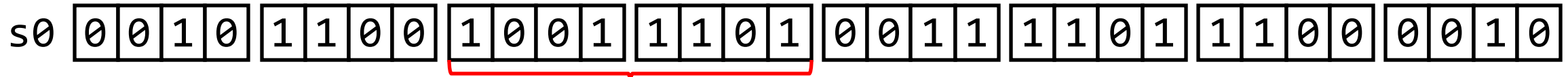
1	0	0	1
---	---	---	---

1	1	0	1
---	---	---	---

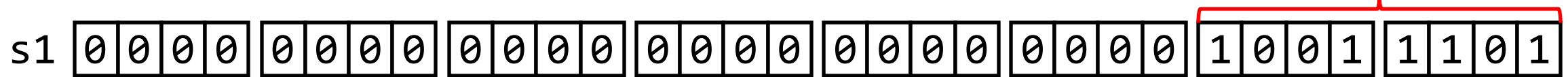


# AND Operations (example) Another Solution

You have:



You want:



First shift s0 to the left to make all unwanted most significant bits zero:

Then shift s0 all the way to the right:




```
slli s1, s0, 8
```

# AND Operations (example) Another Solution

You have:

s0 

0	0	1	0	1	1	0	0	1	0	0	1	1	1	0	1	0	0	1	1	1	1	0	1	1	1	0	0	0	0	1	0
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---



You want:

s1 

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	1	1	0	1
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

First you shift s0 to the left to make all unwanted bits equal zero and put it in s1:

Then shift s1 all the way to the right:

s1 

0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	1	1	0	1	0	0	1	1	1	0	1	1	1	0	0	0	0	1	0
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

```
slli s1, s0, 8
```

```
srli s1, s1, 24
```

# Handling Large Constants

If you try the following instruction in RARS, it will work:

```
li    s0, 0x00FFFF00
```

But the instruction above is a *pseudo* instruction.

To perform the same constant-value load without using a pseudo instruction we can do the following

lui	s0,	0x00FFF	# s0 <- 0x00FFFF000
li	s1,	0xF00	# s1 <- 0x00000F00
or	s0,	s0, s1	# s1 <- 0x00FFFF00

However, RARS does not use this sequence to implement **li**

What does it do instead?

# Large Constants in RARS

How does RARS implement the following pseudo instruction?

```
li    s0, 0x00FFFF000
```

$$\begin{array}{r} 0x0100\ 0000 \\ +\ 0xFFFF\ FF00 \\ \hline 0x00FF\ FF00 \end{array}$$

It replaces li with this:

```
lui   s0, 0x01000
```

```
addi s0, s0, -256          # -256 = 0xF00
```

-256 sign-extended to 32 bits is 0xFFFF FF00

```
s0  0x0000 0000
```

Two instructions instead of three.

```
lui    t0, 0x00FF0
```

# Large Constant Recap

0	0	1	0	1	1	0	0	1	0	0	1	1	1	0	1	0	0	1	1	0	0	0	0	1	0
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

```
lui    t0, 0x00FF0
```

## Large Constant Recap

s0	0	0	1	0	1	1	0	0	1	0	0	1	1	0	1	0	0	1	1	1	1	0	1	1	1	0	0	1	0			
t0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0			
t1	0	0	0	0	0	0	0	0	1	0	0	1	1	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0			
	0	0	0	0	0	0	0	0	1	0	0	1	1	1	0	1	0	0	1	1	1	0	1	1	1	0	0	0	1	0		

```
li      s0, 0x00FFFF000 ➡ lui    s0, 0x01000  
                                addi s0, s0, -256
```