Base64: a binary string is encoded as an ASCII string

- Basic Algorithm:
  - For every three bytes (24 bits) # lcm(6,8) = ?
    - Load and Merge the bytes together
    - Chop and Slide into 4 6-bit chunks
    - Map each 6-bit chunks into a 8-bit ASCII value
    - Store each new the original three bytes with four new bytes
  - Add appropriate padding for remaining bytes
- Mapping ensures the result 8 bits are always printable ASCII characters
- Operations at the assemble level:
  - byte manipulations
  - shifting and masks
- Working at the byte level exposes Endianness

The "encode" subroutine:

```
la $a0, input
la $a1, output
jal encode
```

- The following slides illustrate the steps associated with encoding
  - 24-bits into 4 base64 characters
- The signature (or API) of this subroutine is:
  - void encode(input, output)
    - input: the memory location where the three input values are stored
    - output: the memory location where the four output values are to be stored
- MIPS instructions to call the subroutine:
- MIPS instructions to load and store the input and output within the subroutine

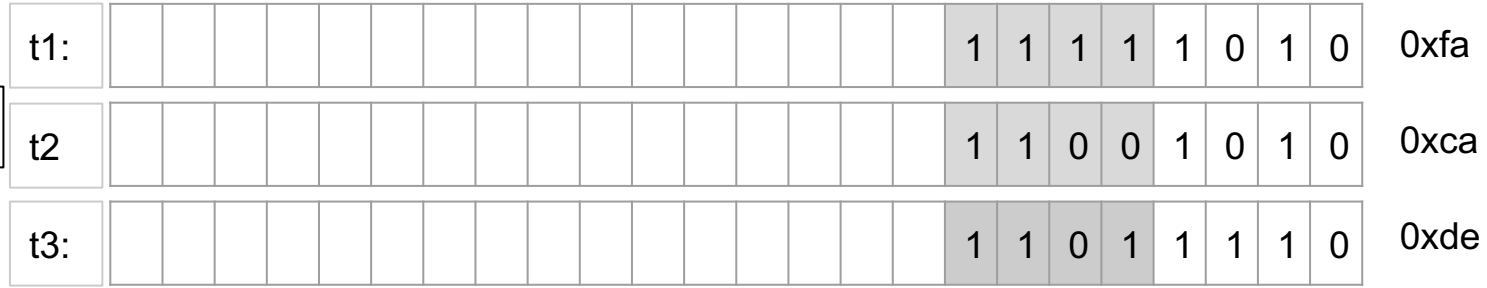
```
# Load 3 input bytes
lbu $t1, 0($a0)
lbu $t2, 1($a0)
lbu $t3, 2($a0)
```

```
# Store 4 output bytes
sb $s1, 0($a1)
sb $s2, 1($a1)
sb $s3, 2($a1)
sb $s4, 3($a1)
```

Load and Merge (shift and meld)

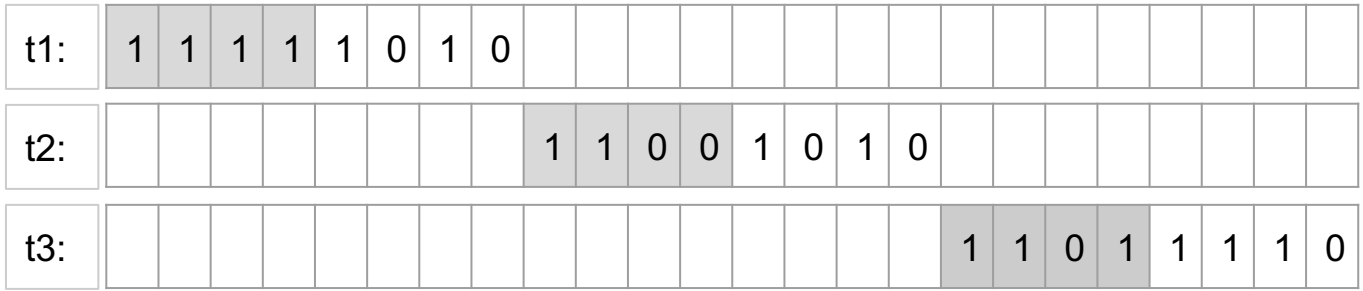
- load:

```
lb $t1, 0($a0)
```

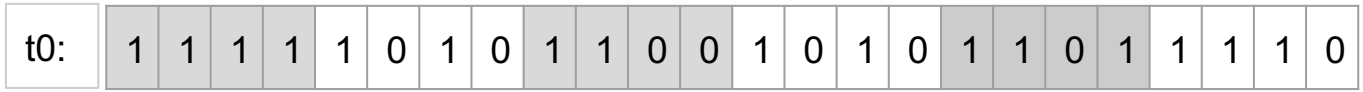


- shift:

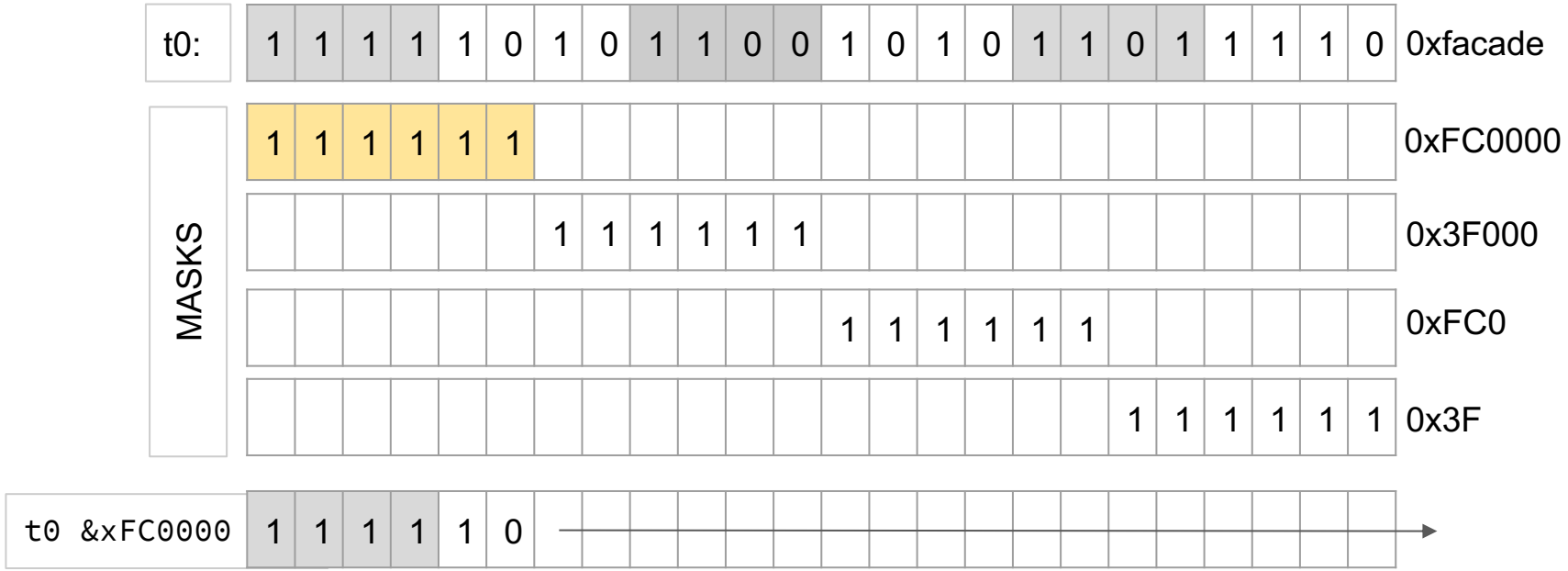
```
sll $t1, $t1, 16
```



- meld:



Chop and Slide:




```
s1= (t0 & xFC0000) >>> 18 == 0x3E
```

```
s2= (t0 & x03F000) >>> 12 == 0x2C
```

```
s3= (t0 & xFC0) >>> 6 == 0x2B
```

```
s4= (t0 & x03F) >>> 0 == 0x1E
```

Mapping:

- Base64 Mapping Table
- Two approaches to mapping:
  - Perform a table lookup
  - Compute the value
    - via the following switch statement
- The computed indices are:
  - `s1 = 0x3E (62)`
  - `s2 = 0x2C (44)`
  - `s3 = 0x2B (43)`
  - `s4 = 0x1E (30)`
- The mapped characters are:
  - '+' (0x2B)
  - 's' (0x73)
  - 'r' (0x72)
  - 'e' (0x65)

```
switch ( index ) {
  0..25  : index += 0 + 'A' ; // A - Z
           break;
  26..51 : index += -26 + 'a'; // a - z
           break;
  52..61 : index += -52 + '0'; // 0 - 9
           break;
  62     : index = '+';
           break;
  63     : index = '/';
                   break;
}
```