Base64

- Extremely common, you use it all the time!
- MIME: Multipurpose Internet Mail Extension
  - Extents the format of email to support other character sets, and ….
  - Used in other contexts as well

- SMTP:
  - cat <binary_file> | mail steve
  - via a modern MSA: mail steve@my.csun.edu
  - Content-Transfer-Encoding: base64
- HTTP/HTML:
  - curl --head $URL
  - Content-Type: text/plain

More Motivation

- What is the content of a text file?
  - cat mytext.file
  - echo 101100101 > ascii_digits.txt
- What is the content of an executable?
  - consider the unprintable characters
- What is the content of a picture?
  - drag and drop a picture

```
$ cat
$ od -t a
$ od -t c
$ od -t d1
$ od -t o2
$ od -t x4
```

- What is the content of data? Individual bits group in bytes
- How can we interpret this data? Any way we want!
- How should we interpret this data? Based upon an agreed upon scheme!

Base64: a binary string is encoded as an ASCII string

```
$ base64 <<< Hello
SGVsbG8K
$ base64 -d  <<<  SGVsbG8K
Hello
```

- A scheme to represent binary data as all "printable" characters
  - Which characters should we use: A-Z, a-z, 0-9: that's 26+26+10 = 62 add + / for 64
  - Hence, we have 2^6 (64) unique characters to use, plus a padding character (=)
- 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

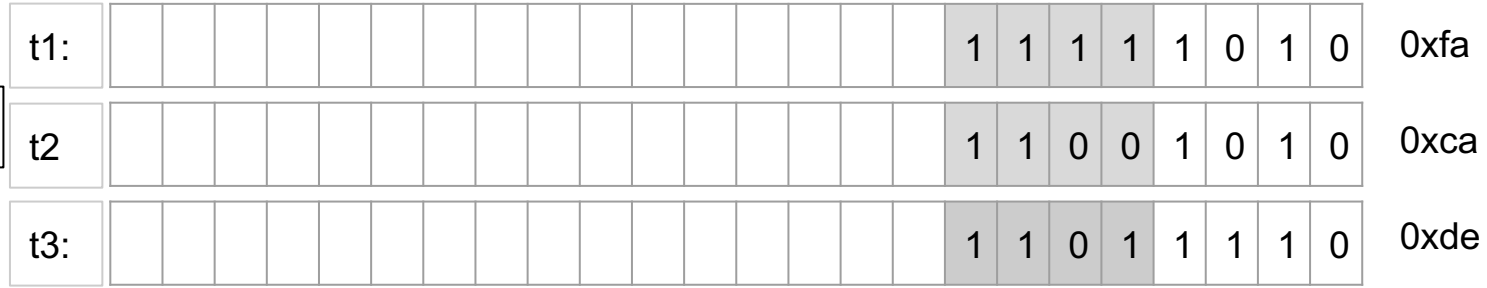
```
# Load 3 input bytes
lbu $t1, 0($a0)
lbu $t2, 1($a0)
lbu $t3, 2($a0)
```

```
# 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)

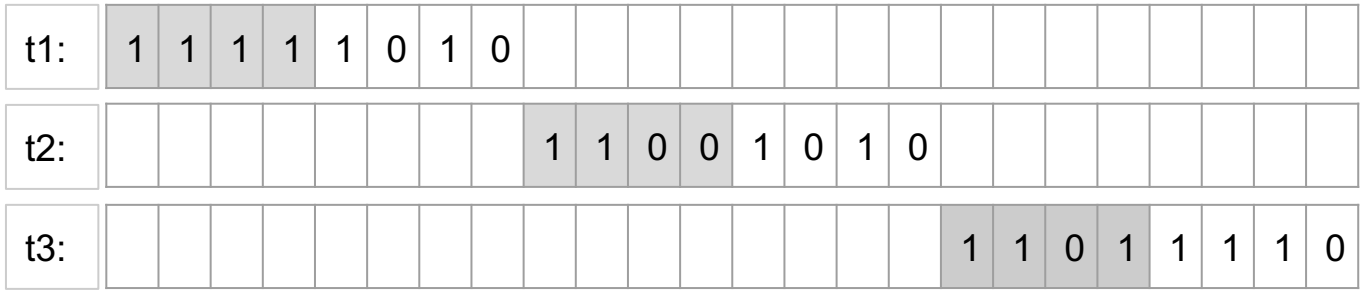
- load:

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

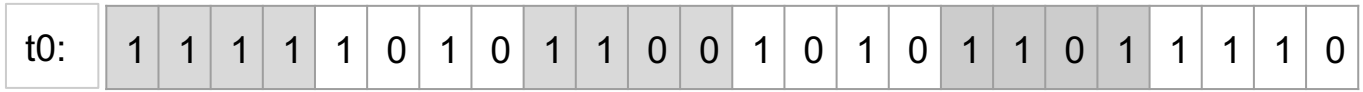


- shift:

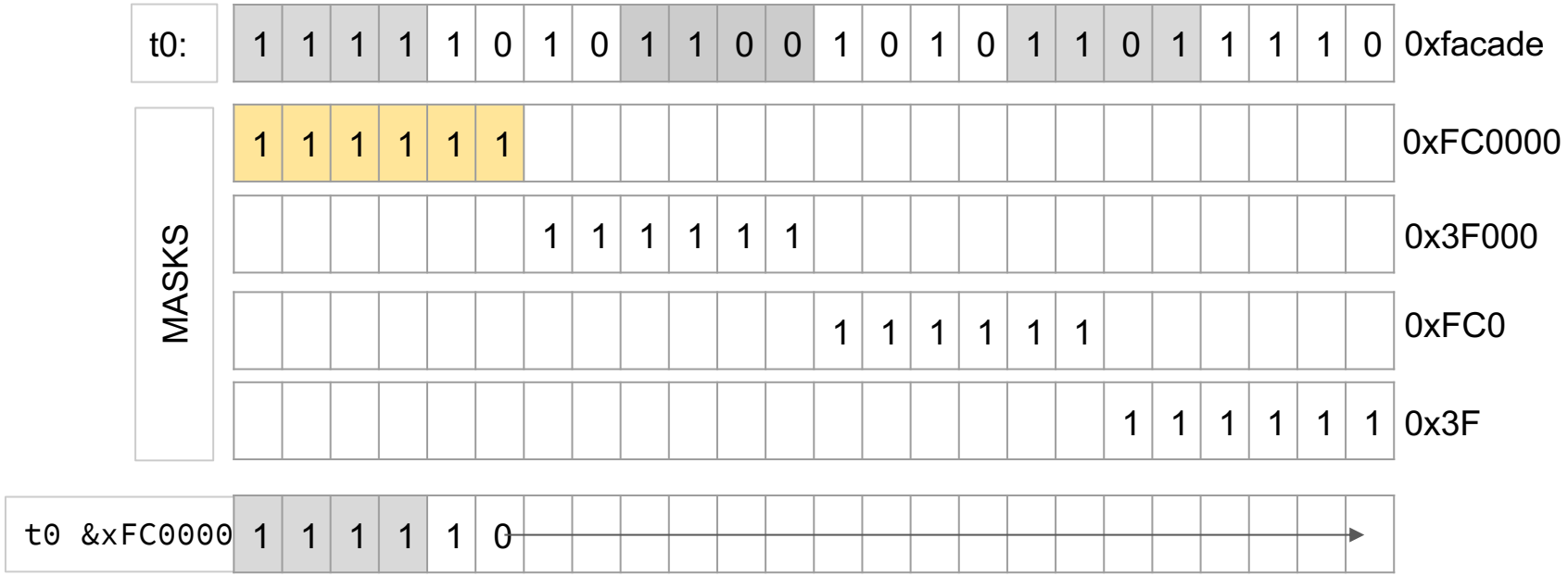
```
sll $t1, $t1, 16
```



- meld:



Chop and Slide:




```
s1= (t0 & xFC0000) >> 18= 11 1110= 0x3E
```

```
s2= (t0 & x03F000) >> 12= 10 1100= 0x2C
```

```
s3= (t0 & xFC0) >> 6 == 10 1011= 0x2B
```

```
s4= (t0 & x03F) >> 0 == 01 1110= 0x1E
```

Mapping:

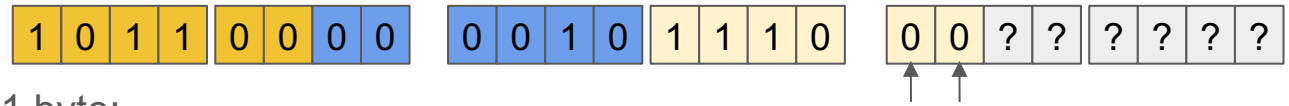
- Base64 Mapping Table
- Two approaches to mapping:
  - Perform a table lookup
  - Compute the value
    - via the following switch statement
- The computed indices are:
  - `s1 = 0x3E (62)`
  - `s2 = 0x2C (44)`
  - `s3 = 0x2B (43)`
  - `s4 = 0x1E (30)`
- The mapped characters are:
  - '+' (0x2B)
  -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

End of File Considerations:

- What if the size of the input is not divisible by 24 # lcm(6,8)
  - You need to pad appropriate number of values to the right
- Remaining 3 bytes:
  - output 4 base64 characters
  - output 0 padding character (=)

- Remaining 2 bytes:
  - output 3 base64 characters (with two zeros as fillers)
  - output 1 padding character (=)



- Remaining 1 byte:
  - output 2 base64 characters (with four zeros as fillers)
  - output 2 padding characters (=)