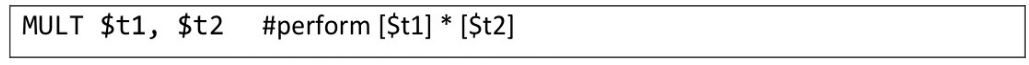
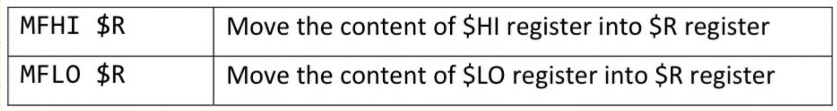
CS2100 - Tutorial 3 - MIPS

Week 5

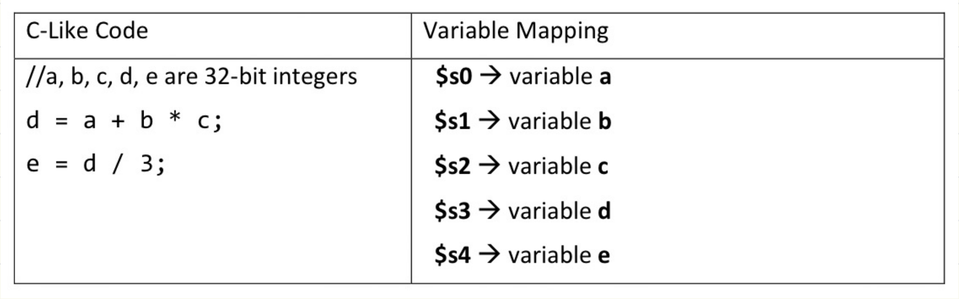
1. MIPS



```
MULT $t1, $t2    #perform [$t1] * [$t2]
```



MFHI $R	Move the content of $HI register into $R register
MFLO $R	Move the content of $LO register into $R register



C-Like Code	Variable Mapping

//a, b, c, d, e are 32-bit integers
d = a + b * c;
e = d / 3;
	$s0 → variable **a**
	$s1 → variable **b**
	$s2 → variable **c**
	$s3 → variable **d**
	$s4 → variable **e**

```
MULT  $s1, $s2
MFLO  $t0
add   $s3, $s0, $t0
addi  $t1, $zero, 3
DIV   $t3, $t1
MFLO  $s4
```

2. Logical Operations   constants: 16-bit

(a). Maximize red color :

```
    lui  $t0, 0x00FF
    or   $s0, $s0, $t0      # Force red bits to 1
```

(b). Invert green colour :

```
    xori $s0, $s0, 0xFF00        # flip green bits
```

(c). Reduce the intensity of blue by half :

```
    andi $t0, $s0, 0x00FF  # Extract blue bits
    srl  $t0, $t0, 1       # Reduce intensity
    srl  $s0, $s0, 8       # Remove blue bits
    sll  $s0, $s0, 8
    or   $s0, $s0, $t0     # Combine
```

3. Code efficiency

(a).

```
# M = M / 2
addi $t2 , $zero, 2
DIV $t1, $t2
MFLO $t1
```

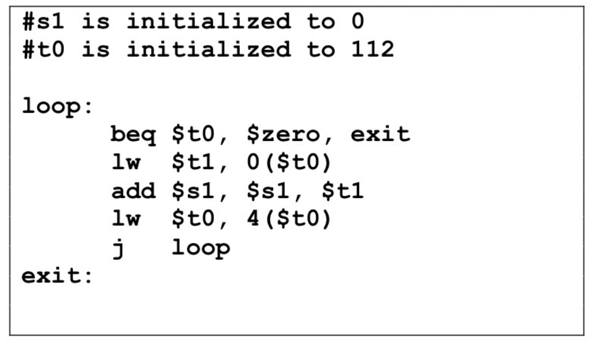
(b).

```
srl $t1, $t1, 1
```

(c). Yes. Fewer instructions to achieve same result ⇒ faster execution

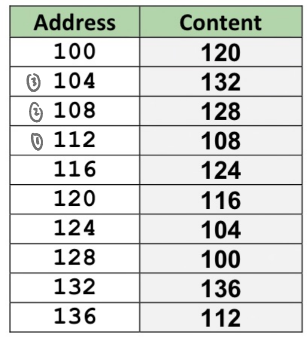
4. Memory instruction and HLL

(a).



```
#s1 is initialized to 0
#t0 is initialized to 112

loop:
      beq $t0, $zero, exit
      lw  $t1, 0($t0)
      add $s1, $s1, $t1
      lw  $t0, 4($t0)
      j   loop
exit:
```



Address	Content
100	120
③ 104	132
② 108	128
① 112	108
116	124
120	116
124	104
128	100
132	136
136	112

$t0 : ~~112~~ ~~108~~ ~~104~~ 116

$t1 : ~~108~~ ~~128~~ 132

$s1 : ~~108~~ ~~236~~ 368

$s1: 368

(b). Address : 120

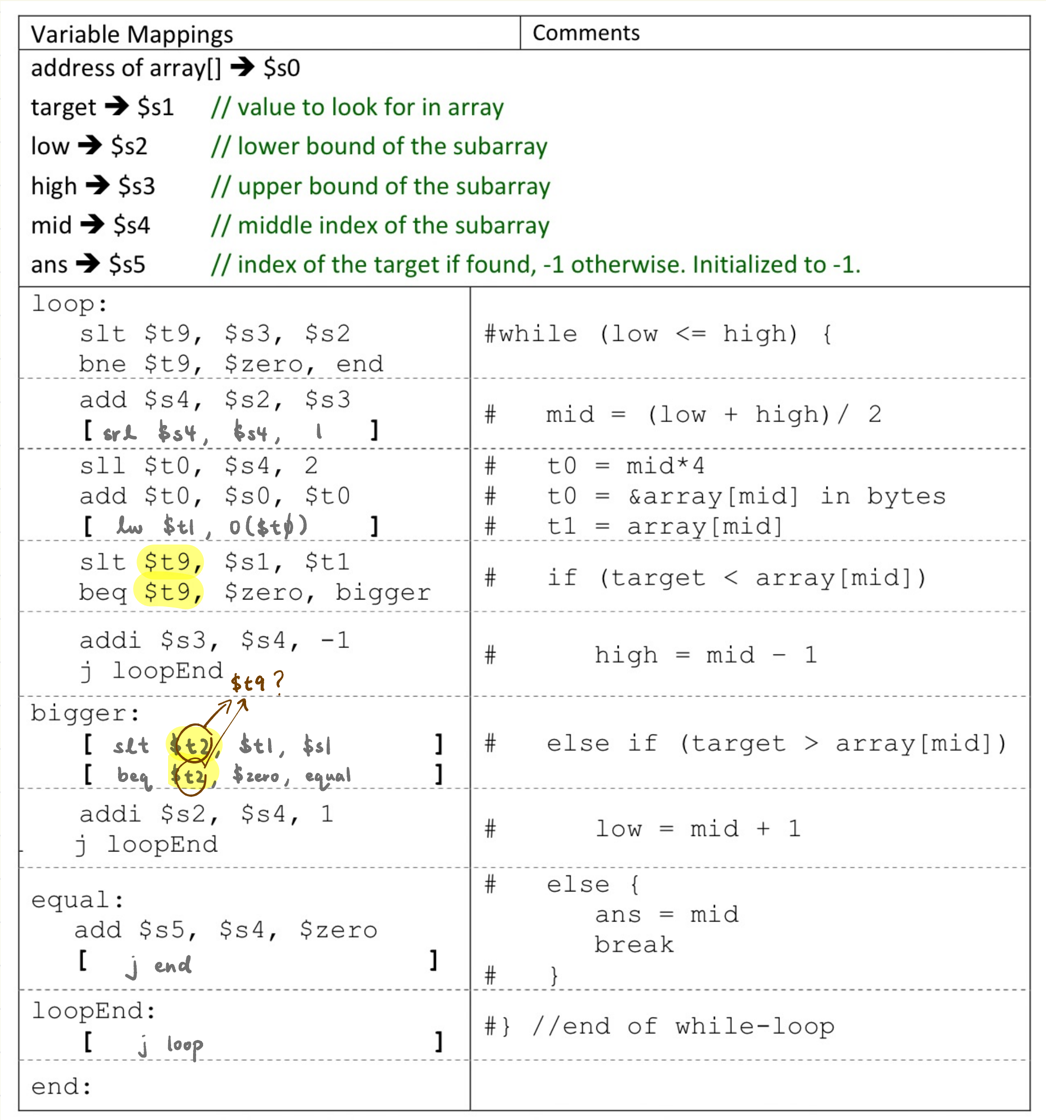
Content : 116 → 0

(c).

```
int s1 = 0;
*int t0 = 112;
int t1;
while ( *t0 != 0 )  {
      t1 = *t0;   // Dereference
      s1 += t1;
      t0 += 4;
}
```

Extra 1. Memory & branches

Binary search



Variable Mappings	Comments
address of array[] ➔ $s0	
target ➔ $s1	// value to look for in array
low ➔ $s2	// lower bound of the subarray
high ➔ $s3	// upper bound of the subarray
mid ➔ $s4	// middle index of the subarray
ans ➔ $s5	// index of the target if found, -1 otherwise. Initialized to -1.
loop: slt $t9, $s3, $s2 bne $t9, $zero, end	#while (low <= high) {
add $s4, $s2, $s3 [srl $s4, $s4, 1]	# mid = (low + high)/ 2
sll $t0, $s4, 2 add $t0, $s0, $t0 [lw $t1, 0($t0)]	# t0 = mid*4 # t0 = &array[mid] in bytes # t1 = array[mid]
slt $t9, $s1, $t1 beq $t9, $zero, bigger	# if (target < array[mid])
addi $s3, $s4, -1 j loopEnd $t9 ?	# high = mid - 1
bigger: [slt $t2, $t1, $s1] [beq $t2, $zero, equal]	# else if (target > array[mid])
addi $s2, $s4, 1 j loopEnd	# low = mid + 1

equal:
add $s5, $s4, $zero
[j end]
	# else {
	ans = mid
	break
	# }

loopEnd: [j loop]	#} //end of while-loop
end:	