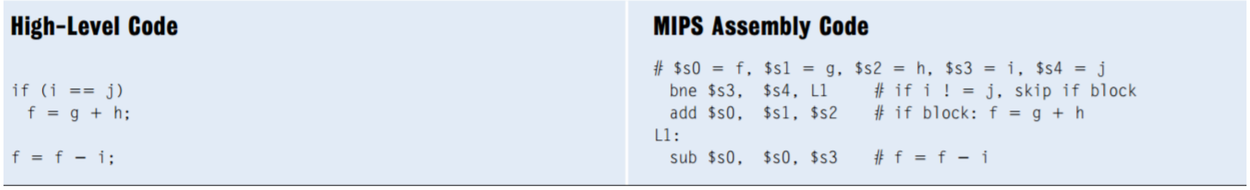
MIPS Homework 02.
Conditions, loops and arrays.

High-level code to MIPS

if STATEMENT



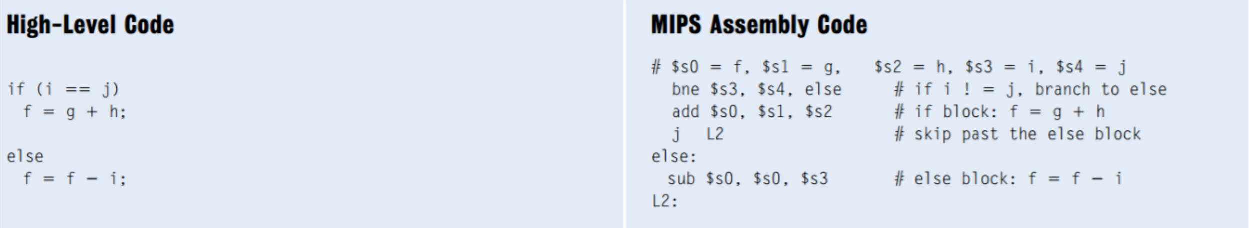
High-Level Code

```
if (i == j)
  f = g + h;

f = f – i;
```

MIPS Assembly Code

```
# $s0 = f, $s1 = g, $s2 = h, $s3 = i, $s4 = j
  bne $s3,  $s4, L1     # if i ! = j, skip if block
  add $s0,  $s1, $s2    # if block: f = g + h
L1:
  sub $s0,  $s0, $s3    # f = f – i
```

If/else STATEMENT



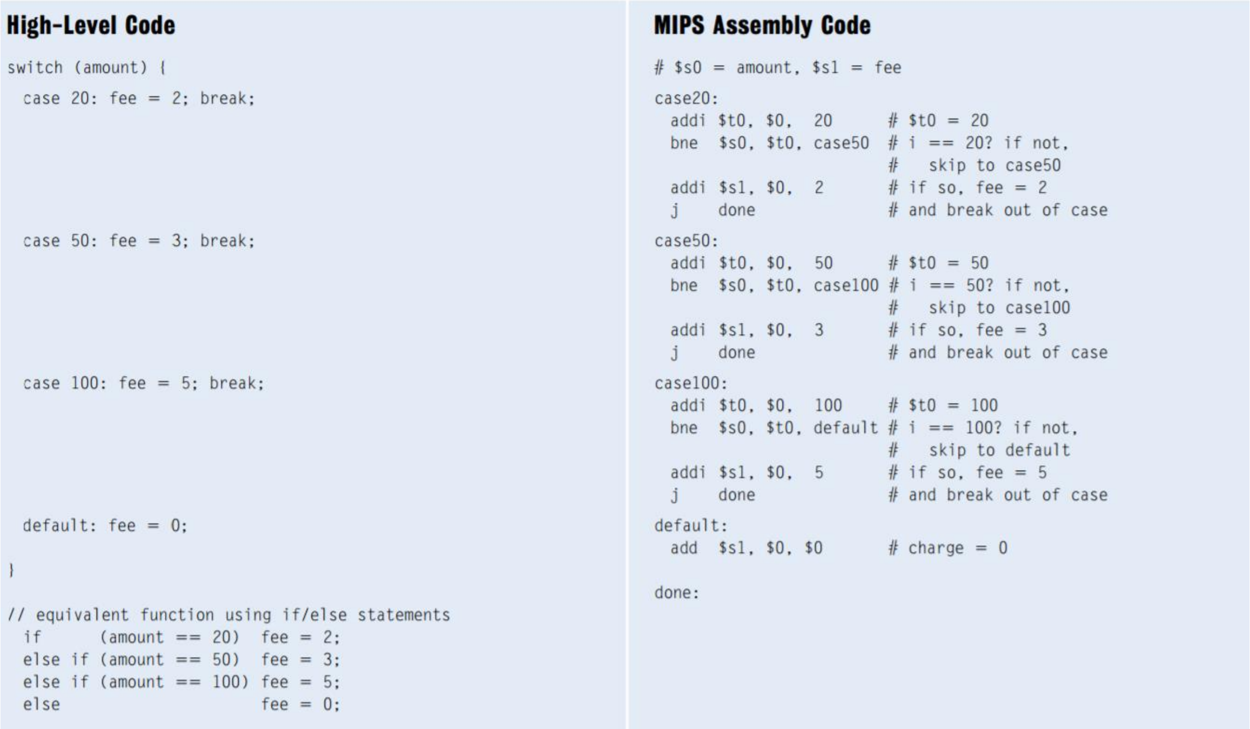
High-Level Code

```
if (i == j)
  f = g + h;

else
  f = f – i;
```

MIPS Assembly Code

```
# $s0 = f, $s1 = g,  $s2 = h, $s3 = i, $s4 = j
  bne $s3, $s4, else     # if i ! = j, branch to else
  add $s0, $s1, $s2      # if block: f = g + h
  j   L2                 # skip past the else block
else:
  sub $s0, $s0, $s3      # else block: f = f – i
L2:
```

switch/case STATEMENT



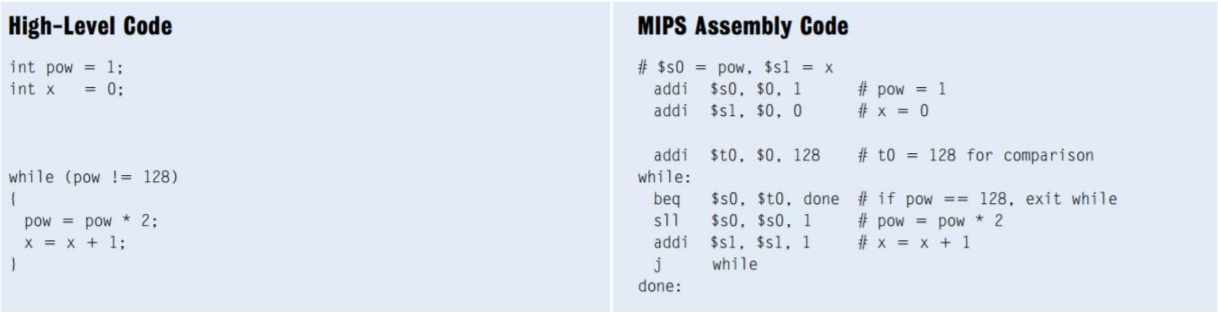
High-Level Code

```
switch (amount) {
  case 20: fee = 2; break;

  case 50: fee = 3; break;

  case 100: fee = 5; break;

  default: fee = 0;

}

// equivalent function using if/else statements
  if      (amount == 20)  fee = 2;
  else if (amount == 50)  fee = 3;
  else if (amount == 100) fee = 5;
  else                    fee = 0;
```

MIPS Assembly Code

```
# $s0 = amount, $s1 = fee
case20:
  addi $t0, $0,  20       # $t0 = 20
  bne  $s0, $t0, case50   # i == 20? if not,
                          #   skip to case50
  addi $s1, $0,  2        # if so, fee = 2
  j    done               # and break out of case
case50:
  addi $t0, $0,  50       # $t0 = 50
  bne  $s0, $t0, case100  # i == 50? if not,
                          #   skip to case100
  addi $s1, $0,  3        # if so, fee = 3
  j    done               # and break out of case
case100:
  addi $t0, $0,  100      # $t0 = 100
  bne  $s0, $t0, default  # i == 100? if not,
                          #   skip to default
  addi $s1, $0,  5        # if so, fee = 5
  j    done               # and break out of case
default:
  add  $s1, $0, $0        # charge = 0

done:
```

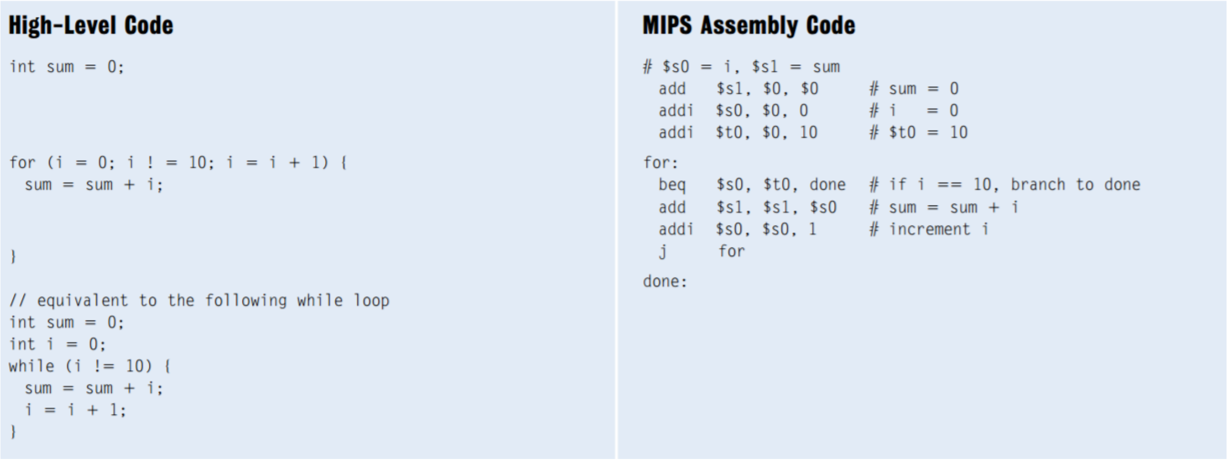
while LOOP



High-Level Code	MIPS Assembly Code

```
int pow = 1;
int x   = 0;


while (pow != 128)
{
  pow = pow * 2;
  x = x + 1;
}
```
	```
	# $s0 = pow, $s1 = x
	  addi  $s0, $0, 1      # pow = 1
	  addi  $s1, $0, 0      # x = 0

	  addi  $t0, $0, 128    # t0 = 128 for comparison
	while:
	  beq   $s0, $t0, done  # if pow == 128, exit while
	  sll   $s0, $s0, 1     # pow = pow * 2
	  addi  $s1, $s1, 1     # x = x + 1
	  j     while
	done:
	```

for LOOP



High-Level Code	MIPS Assembly Code

```
int sum = 0;



for (i = 0; i ! = 10; i = i + 1) {
  sum = sum + i;


}

// equivalent to the following while loop
int sum = 0;
int i = 0;
while (i != 10) {
  sum = sum + i;
  i = i + 1;
}
```
	```
	# $s0 = i, $s1 = sum
	  add   $s1, $0, $0     # sum = 0
	  addi  $s0, $0, 0      # i   = 0
	  addi  $t0, $0, 10     # $t0 = 10
	for:
	  beq   $s0, $t0, done  # if i == 10, branch to done
	  add   $s1, $s1, $s0   # sum = sum + i
	  addi  $s0, $s0, 1     # increment i
	  j     for
	done:
	```

Example:

The following high-level code adds the powers of 2 from 1 to 100. Translate it into assembly language.

```
        // high-level code

        int sum = 0;

        for (i = 1; i < 101; i = i * 2)

                sum = sum + i;
```

Solution:

The assembly language code uses the set less than (slt) instruction to perform the less than comparison in the for loop.

```
        # MIPS assembly code

        # $s0 = i, $s1 = sum
```

```
addi $s1, $0, 0 # sum = 0

addi $s0, $0, 1 # i = 1

addi $t0, $0, 101 # $t0 = 101

loop: slt $t1, $s0, $t0 # if (i < 101) $t1 = 1, else $t1 = 0

beq $t1, $0, done # if $t1 == 0 (i >= 101), branch to done

add $s1, $s1, $s0 # sum = sum + i

sll $s0, $s0, 1 # i = i * 2

j loop

done:
```

Exercise 01.

Implement the following high-level code segments using the slt instruction. Assume the integer variables g and h are in registers $s0 and $s1, respectively.

(a)

```
if (g > h)
        g = g + h;
else
        g = g - h;
```

(b)

```
if (g >= h)
        g = g + 1;
else
        h = h - 1;
```

(c)

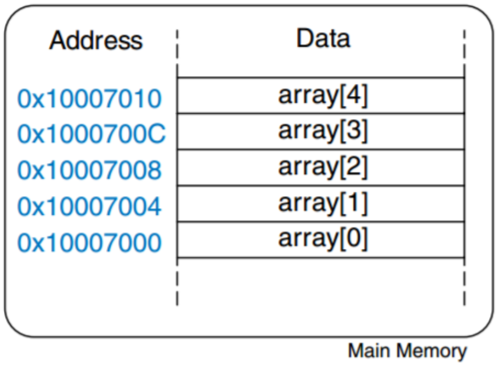
```
if (g <= h)
        g = 0;
else
        h = 0;
```

Exercise 02:

Each number in the Fibonacci series is the sum of the previous two numbers. Write a procedure called fib in a high-level language that returns the Fibonacci number for any nonnegative value of n. Use a loop. Clearly comment your code. Add comments after every line of code that explain clearly what it does. Write MIPS assembly. Use the MARS simulator to test your code on fib(9).

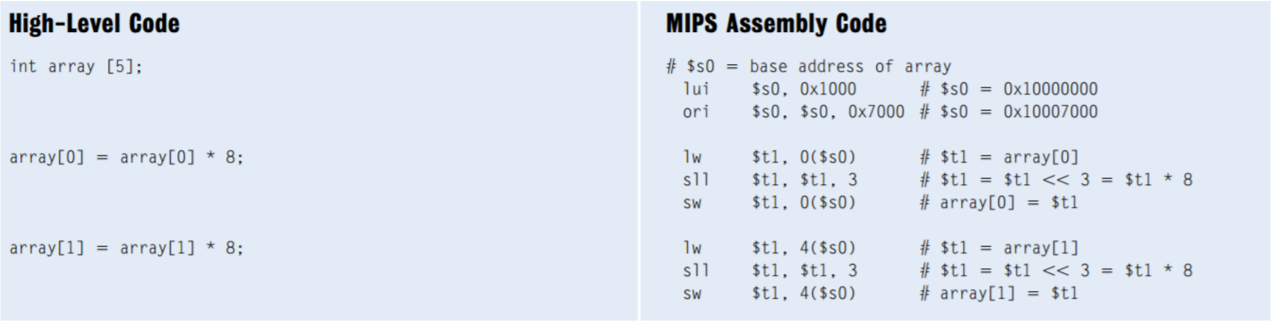
Arrays

Allocation of arrays in the memory (base address is 0x1000700):



Address	Data
0x10007010	array[4]
0x1000700C	array[3]
0x10007008	array[2]
0x10007004	array[1]
0x10007000	array[0]

How to access:



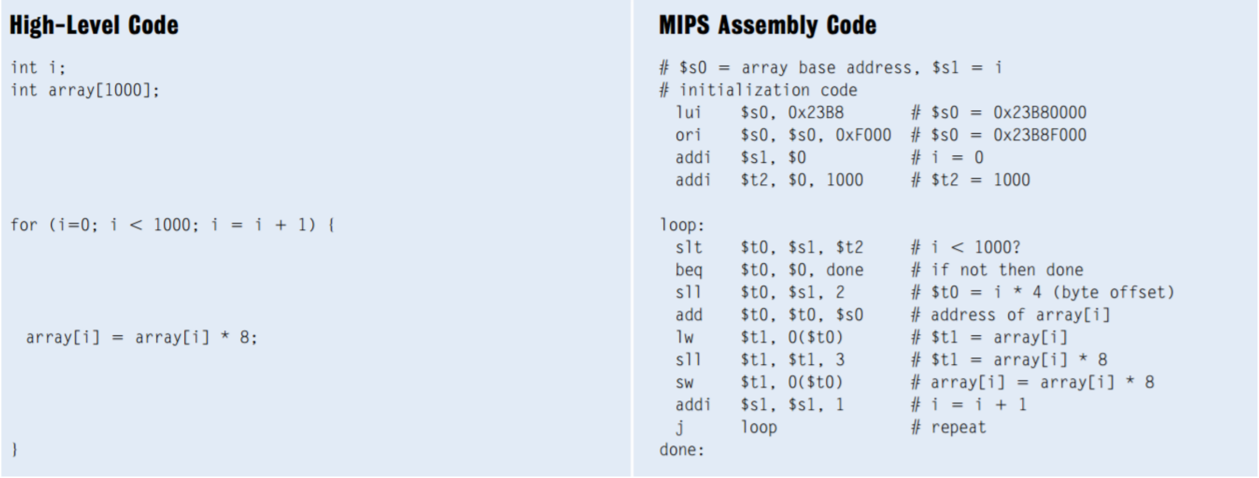
High-Level Code

```
int array [5];


array[0] = array[0] * 8;


array[1] = array[1] * 8;
```

MIPS Assembly Code

```
# $s0 = base address of array
  lui    $s0, 0x1000       # $s0 = 0x10000000
  ori    $s0, $s0, 0x7000  # $s0 = 0x10007000

  lw     $t1, 0($s0)       # $t1 = array[0]
  sll    $t1, $t1, 3       # $t1 = $t1 << 3 = $t1 * 8
  sw     $t1, 0($s0)       # array[0] = $t1

  lw     $t1, 4($s0)       # $t1 = array[1]
  sll    $t1, $t1, 3       # $t1 = $t1 << 3 = $t1 * 8
  sw     $t1, 4($s0)       # array[1] = $t1
```

Accessing array with loop with the help of logical shifts:



High-Level Code

```
int i;
int array[1000];





for (i=0; i < 1000; i = i + 1) {




  array[i] = array[i] * 8;





}
```

MIPS Assembly Code

```
# $s0 = array base address, $s1 = i
# initialization code
  lui    $s0, 0x23B8        # $s0 = 0x23B80000
  ori    $s0, $s0, 0xF000   # $s0 = 0x23B8F000
  addi   $s1, $0            # i = 0
  addi   $t2, $0, 1000      # $t2 = 1000

loop:
  slt    $t0, $s1, $t2      # i < 1000?
  beq    $t0, $0, done      # if not then done
  sll    $t0, $s1, 2        # $t0 = i * 4 (byte offset)
  add    $t0, $t0, $s0      # address of array[i]
  lw     $t1, 0($t0)        # $t1 = array[i]
  sll    $t1, $t1, 3        # $t1 = array[i] * 8
  sw     $t1, 0($t0)        # array[i] = array[i] * 8
  addi   $s1, $s1, 1        # i = i + 1
  j      loop               # repeat
done:
```

Example:

The following high-level code converts a ten-entry array of characters from lower-case to upper-case by subtracting 32 from each array entry. Translate it into MIPS assembly language. Remember that the address difference between array elements is now 1 byte, not 4 bytes. Assume that $s0 already holds the base address of chararray.

```
// high-level code
char chararray[10];
int i;
for (i = 0; i != 10; i = i + 1)
    chararray[i] = chararray[i] - 32;
```

Solution:

```
# MIPS assembly code
# $s0 = base address of chararray, $s1 = i
addi $s1, $0, 0 # i = 0
addi $t0, $0, 10 # $t0 = 10
loop: beq $t0, $s1, done # if i == 10, exit loop
add $t1, $s1, $s0 # $t1 = address of chararray[i]
lb $t2, 0($t1) # $t2 = array[i]
addi $t2, $t2, -32 # convert to upper case: $t1 = $t1 - 32
sb $t2, 0($t1) # store new value in array: chararray[i] = $t1
addi $s1, $s1, 1 # i = i + 1
j loop # repeat
done:
```

Exercise 03:

Write a procedure in a high-level language for *int find42(int array[], int size)*. size specifies the number of elements in the array. array specifies the base address of the array. The procedure should return the index number of the first array entry that holds the value 42. If no array entry is 42, it should return the value -1. Convert the high-level procedure into MIPS assembly code.

Exercise 04:

The high-level procedure strcpy copies the character string x to the character string y.

```
// high-level code
void strcpy(char x[], char y[])
{
    int i 0;
    while (x[i] != 0)
    {
        y[i] = x[i];
        i = i + 1;
    }
}
```

Implement the strcpy procedure in MIPS assembly code. Use $s0 for i.