CLASS MATERIAL – EXERCISE 0

```
int numbers[3] = {10,12,14};
int answer = 0;
if (numbers[2] == 0){
      answer = 1;
}
else{
      answer = 0
}
```

—— Run these assembly code below on BSIM:

```
.include beta.uasm

PUSH(R1)
PUSH(R2)

CMOVE(numbers, R1)
CMOVE(2, R2)

MULC(R2, 4, R2)
ADD(R1, R2, R2)
LD(R2, 0, R1)
BEQ(R1, branch_true, R31)
CMOVE(0, R0)
ST(R0,answer)

exit:
POP(R2)
POP(R1)
HALT()

branch_true:
CMOVE(1, R0)
ST(R0, answer)
BR(exit)


answer : LONG(0)
numbers: LONG(10)
LONG(12)
LONG(14)
```

CLASS MATERIAL – EXERCISE 1

```
int multiply(int a, int b){
      int answer = [4]; what is the C-code here?
      return answer;
}

int add(int a, int b){
      int answer = a + b;
      return answer;
}

int main(){
      int res = multiply(2,5);
      res = add(res, res);
}
```

—— Run these assembly code below on BSIM:

```
res = 0x120  | NAMING ADDRESS 0X120 AS RES
. = 0x0000
ALLOCATE(90) | Changed from 100 to 90 because b-sim's memory is small
CMOVE(2, R1)
CMOVE(5, R2)
PUSH(R2)
PUSH(R1)
BR(multiply, LP)
DEALLOCATE(2)
PUSH(R0)
PUSH(R0)
BR(add, LP)
DEALLOCATE(2)
ST(R0, res)
DEALLOCATE(90) | Changed from 100 to 90 because b-sim's memory is small
HALT()


multiply: PUSH(LP)
PUSH(BP)
MOVE(SP, BP)
PUSH(R3)
PUSH(R4)
[1](BP, -12, R3)  what is the instruction here?
[2](BP, -16, R4)  what is the instruction here?
MUL(R3, R4, R0)
POP(R4)
POP(R3)
MOVE(BP, SP)
POP(BP)
POP(LP)
JMP(LP)

add: PUSH(LP)
PUSH(BP)
MOVE(SP, BP)
```

```
PUSH(R3)
PUSH(R4)
LD(BP,-12, R3)
LD(BP,-16, R4)
```
[3] -- what is the assembly instruction here?
```
POP(R4)
POP(R3)
MOVE(BP, SP)
POP(BP)
POP(LP)
JMP(LP)
```

Assume initially all regs content is zero. The program is paused and current stack content is:

```
      0X00000002
      0X00000005
      0X00000020
      0X00000000
      0X00000000            <- BP is pointing here
      0X00000000
SP->
```

Fill up the blanks [1] to [4] above and answer the questions below.
[1], [2] : LD
[3] : ADD(R3, R4, R0)
[4] : a+b

1: Where does the stack frame begins?
Memory address: 0x168
The initial content of SP is 0. When instruction ALLOCATE(90) is executed, it increases the content of SP by 90 * 4 = 360. Therefore, the stack starts at address 0x168 (in hex).

2. What is the content of BP currently, at this stack state?
The content of BP is 0x178.

3. This breakpoint pauses the code when it is about to execute which beta instruction(s)?
Either of these 3: LD(BP, -12, R3), or LD(BP, -16, R4), or MUL(R3, R4, R0)

CLASS MATERIAL – EXERCISE 2

```
/* calculates the floor(log2(n)) */
int floor_log(int n)
{
      if(n == 1)
         return 0;
      else
       return 1 + [5] -- what is the C-code here?
}

int main(){
      int answer = floor_log(10);
      return answer;
}
```

```
.include beta.uasm

answer = 0x104
ALLOCATE(100)
CMOVE(8,R1)
CMOVE(0,R0)

PUSH(R1)
BR(floor_log, LP)
DEALLOCATE(1)

ST(R0, answer)
DEALLOCATE(100)
HALT()

floor_log: PUSH(LP)
PUSH(BP)
MOVE(SP, BP)

PUSH(R1)
PUSH(R2)
CMOVE(1,R2)
LD(BP,-12,R1)
[1]
BNE(R2, exit_sequence, R31)
DIVC(R1, 2, R1)

PUSH(R1)
BR(floor_log, LP)
DEALLOCATE(1)
ADDC(R0, 1, R0)


exit_sequence:
POP(R2)
POP(R1)
MOVE([2], [3])
POP(BP)
POP(LP)
[4]
```

Inspect stack:

```
      0x00000008      R1 -- this is at 0x00000190 (100*4) due to ALLOCATE
      0x00000018      LP(from main)
      0x00000000      BP
      0x00000008      R1
      0x00000000      R2
      0x00000004      R1 (new argument)
      0x0000006C      LP (from 1st rec)
      [6]             BP (from 1st rec)
      0x00000004      R1
      0x00000000      R2
      0x00000002      R1 (new argument, prepping for 2nd rec call)
      0x0000006C      LP (from 2nd rec)
SP—>
```

Fill up the blanks [1] to [6] above and answer the questions below.

[1] : **CMPEQ(R1, R2, R2)**
[2],[3] : **BP, SP**
[4] : **JMP(LP)**
[5] : **floor_log(n/2)**
[6] : **0x0000019C**

1. **At which assembly instruction is this breakpoint at? PUSH(BP)**
2. **How many stack frame used at max per function call? 5 words: LP, BP, R1, R2, and the new argument**
3. **How many calls of floor_log has been made? 3 calls. When the program is currently paused, we are in the middle of the 3rd function call.**
4. **At which address is SP now? Address 448 : 0x000001C0, we can count that there are 12 lines after the stack frame begins. The stack frame begins at address 400: that is 0x00000190.**

CLASS MATERIAL – EXERCISE 3

```
int sum_array(int* array_name, int index){
      int answer = 0;
      for (int i=0; i<index; i++){
            answer += array_name[i];
      }
      return answer;
}
int main(){
      int array[5] = {1,2,3,4,5};
      answer = sum_array(array, 3);
      return answer;
}
```

```
.include beta.uasm
.=0x00000000
ALLOCATE(100)
PUSH(R1)
PUSH(R2)
CMOVE(array, R1)
CMOVE(3, R2)
PUSH(R2)
PUSH(R1)
BR(sum_array, LP)
DEALLOCATE(2)
POP(R2)
POP(R1)
ST(R0, answer)
HALT()

sum_array: PUSH(LP)
PUSH(BP)
MOVE(SP, BP)

PUSH(R1)
PUSH(R2)
PUSH([1])
PUSH([2])
LD(BP,-12, [3])
LD(BP,-16, [4])

CMOVE(0, R3)
CMOVE(0, R0)


beginfor_loop: CMPEQ(R3, R2, R4)
BEQ(R4, bodyfor_loop, LP)
POP(R4)
POP(R3)
POP(R2)
POP(R1)

MOVE(BP, SP)
POP(BP)
```

```
POP(LP)
JMP(LP)

bodyfor_loop: MULC(R3, 4, R4)
ADD(R1, R4, R4)
LD(R4, 0, R4)
[5](R0, R4, R0)
ADDC(R3, 1, R3)
BR(beginfor_loop)

answer : LONG(0)
array: LONG(1)
LONG(2)
LONG(3)
LONG(4)
LONG(5)
```

Fill up the blanks [1] to [5] above and answer the questions below.

[1], [2]: R3, R4
[3]: R1
[4]: R2
[5]: ADD

1. What is the address of "array"? 0x000000EC (59 lines of instructions/ content above "array")

2. What is the address of the items in "array"?
   1. Address 0x000000EC contains 0x00000001
   2. Address 0x000000F0 contains 0x00000002
   3. Address 0x000000F4 contains 0x00000003
   4. Address 0x000000F8 contains 0x00000004
   5. Address 0x000000FC contains 0x00000005

3. What is stored at answer when the entire program is executed? 6
4. What is the value of LP after BR(sum_array, LP) is just executed? 0x00000030
5. What is the max size of the stack frame of sum_array? 6
6. Which register contains the *index of the array to access?* R3
7. Which instruction line access the array value? LD(R4,0,R4)
8. What does R4 contain and what's its purpose in the code? The absolute address of the item so that we know which item of "array" we should get.