

# Tutorial 2: Programmable Model of Computer

## Problem 1: C Functions

Consider the C code in Figure 1. What choice for the blank ensures that `7 = 7` is printed?

```
#include <stdio.h>

int *f (int varNotToSet, int varToSet) {
    int n = 7;
    varToSet = 7;
    return ____;
    /* &n, &varNotToSet, or &varToSet could go here */
}

int main ( ) {
    int *ptr;
    int k1 = 7, k2 = 0;
    ptr = f(k1, k2);
    printf ("7 = ");
    printf ("%d\n", *ptr);
    return 0;
}
```

Figure 1: Program on Data Type Conversion

None of the three options `&n`, `&varNotToSet`, or `&varToSet` can produce `(7 = 7)`. All the three variables are set to value 7 inside the function `f`, and the address of which ever variable that is placed in `(----)` is returned to the `main` function. However, the addresses of all the three variables are placed in a temporary area of memory called “*stack*”. The stack is given to function `f` when it is called, taken away when it returns, and given to the next function `printf`. That means a call to function `printf` will overwrite the stack area that was occupied by the function `f`. So, when it uses the `*ptr` to access the content of the variable placed in `(----)`, it will not find the value 7. What it will find is the value placed in that memory location by the function `printf`.

This problem can be rectified by declaring any of the variables `&n`, `&varNotToSet`, and `&varToSet` as `static` and placing that variable in `(----)`. This will ensure that the memory location is allocated in the permanent “*static*” area.

## Problem 2: ARM Programmer’s Architecture

Consider the ARM Processor architectures. Answer the following questions.

How many registers are visible to the programmer?

What is the size of these registers?

If the content of register R2 is 134, draw a picture to show the placement of the number 134 in the register bits.

The ARM processor has 16 registers (R0 – R15) visible to the programmer, with each register 32 bits. The register bits are numbered 0 to 31.

The binary equivalent of 134 is `b1000,0110`. Placement of 134 in register R2 is shown in Figure 2.

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	1	1	0	
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	10	09	08	07	06	05	04	03	02	01	00

Figure 2: The placement of 134 in register R2

### Problem 3: Data Manipulation Instructions

Consider the assembly code in Figure 3.

Provide a comment for each line assembly instruction.

If register R2 contains 23, what are the contents of registers R1 – R4 after the execution all the assembly instructions?

```
mov r1, r2
add r3, r1, r2
and r4, r3, r2
orr r4, r3, r4
```

Figure 3: Program on arithmetic instructions

```
mov r1, r2      ; r1 ← r2, r1=23, r2=23
add r3, r1, r2  ; r3 ← r1 + r2, r3 = 23 + 23 = 46
and r4, r3, r2  ; r4 ← r3 & r2 (bitwise ANDing),
                ; r4 = (46=0b10,1110) AND (23=0b01,0111) = 00,00110 = 6
orr r4, r3, r4  ; r4 ← r3 | r4 (bitwise Oring),
                ; r4 = (46=0b10,1110) OR (6=0b00,0110) = 0b10,1110 = 46
```

Figure 4: Program on arithmetic instructions with comments

### Problem 4: Load Store Instruction

Consider the assembly code in Figure 5.

Provide a comment for each line assembly instruction.

If register R2 contains 100, and content of memory at locations 100 and 104 are 45 and 1004, what are the contents of registers R1 – R3 and memory locations 100 and 104 after the execution all the assembly instructions?

```
ldr r1, [r2]
ldr r2, [r2+4]
add r3, r1, r2
str r3, [r2 + 4]
```

Figure 5: Program on load store instructions

```
ldr r1, [r2]    ; r1 ← mem[r2], r1 ← mem[100], r1 = 45
ldr r2, [r2+4]  ; r2 ← mem[r2 + 4], r2 ← mem[104], r2 = 1004
add r3, r1, r2  ; r3 ← r1 + r2, r3 = 45 + 1004 = 1049
str r3, [r2 + 4] ; r3 → mem[r2 + 4], r3 → mem[1008], mem[1008] = 1049
```

Figure 6: Program on load store instructions with comments