Lab 7 – NIOS II Stack and Function calls

**Deliverables & Questions Worksheet –** *See instructions for context*

**Name & Date & Section:**

Sophia Sun & Oct 12, 2022 & Section A

**D1: C-Code for Factorial Function**

int factorial(int n) {

int value = n;

if (n == 0) {

value = 1;

} else {

value = n \* factorial(n – 1);

}

return value;

}

**Q2: “How does it know when to break out of the loop?”**

The program checks if the value stores in r16 is equal to the value stores in r18 at the end of each loop, and if not so, it will branch the program back to the instruction before the loop, and the value stores in r16 will be incremented during the loop, and the program will check again after the loop until r16 = r18.

**D3: Main Loop Table**

|  |  |  |
| --- | --- | --- |
| **Main() Iteration Loop Count (“n”)** | **R4 Value Before “Call Factorial”** | **R2 Value After “Call Factorial”** |
| 0 | 0 | 1 |
| 1 | 1 | 1 |
| 2 | 2 | 2 |
| 3 | 3 | 6 |
| 4 | 4 | 24 |
| 5 | 5 | 120 |
| 6 | 6 | 720 |
| 7 | 7 | 5040 |
| 8 | 8 | 40320 |
| 9 | 9 | 362880 |
| 10 | 10 | 3628800 |

**Q4: “Describe in words how registers 2 and 4 are being used in the main routine.”**

R4 stores the value that is going to multiply with the value on r2 in the current loop, and r2 stores the result of the multiplication. When the loop ends, the value on r2 is the result of the factorial function.

**D5: Recursion Analysis Table**

\* Note each of the 4 column headers describes one of the breakpoints. Every row should have a value in only the column associated with the current breakpoint’s description.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Breakpoint Count** | **Value of n in Main()** | **Location of Breakpoint (one per row)** | | | |
| **R4 Value at main() call statement** | **R2 Value After main() call statement** | **R4 Value at factorial() call statement** | **R2 Value After factorial() call statement** |
| 1 | 0 | 0 |  |  |  |
| 2 | 0 |  | 1 |  |  |
| 3 | 1 | 1 |  |  |  |
| 4 | 1 |  |  | 0 |  |
| 5 | 1 |  |  |  | 1 |
| 6 | 1 |  | 1 |  |  |
| 7 | 2 | 2 |  |  |  |
| 8 | 2 |  |  | 1 |  |
| 9 | 1 |  |  | 0 |  |
| 10 | 1 |  |  |  | 1 |
| 11 | 2 |  |  |  | 1 |
| 12 | 2 |  | 2 |  |  |
| 13 | 3 | 3 |  |  |  |
| 14 | 3 |  |  | 2 |  |
| 15 | 2 |  |  | 1 |  |
| 16 | 1 |  |  | 0 |  |
| 17 | 1 |  |  |  | 1 |
| 18 | 2 |  |  |  | 1 |
| 19 | 3 |  |  |  | 2 |
| 20 | 3 |  | 6 |  |  |

**Q6: “Comment on the sequence of events listed in the table and what they represent.”**

When the loop starts, the initial value of r4 and r2 will be passed into the factorial() function; the function will runs, and the result will be passed back to the main() function. When the value of r4 is greater than 1, the factorial will recursively calculate the value of factorial(r4 – 1), factorial(r4 – 2)… and pass the product of all values back to the main() function.

**D7: Global Array Table**

|  |  |  |
| --- | --- | --- |
| **Main Loop Iteration (“n”)** | **Effective Address Used by “stw” (use hexadecimal)** | **Data to be stored at that address (use decimal)** |
| 0 | 0x000047cc | 1 |
| 1 | 0x000047d0 | 1 |
| 2 | 0x000047d4 | 2 |
| 3 | 0x000047d8 | 6 |
| 4 | 0x000047dc | 24 |
| 5 | 0x000047e0 | 120 |
| 6 | 0x000047e4 | 720 |
| 7 | 0x000047e8 | 5040 |
| 8 | 0x000047ec | 40320 |
| 9 | 0x000047f0 | 362880 |
| 10 | 0x000047f4 | 3628800 |

**D8: Factorial Array Memory Screenshot**

图形用户界面, 应用程序

中度可信度描述已自动生成

**Q9: “How many 4-byte (1-word) elements are currently on the stack?”**

The address: 0x00007ffc

Only one value, which is 0.

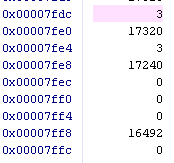
**Q10: “Identify the address pointed to by the stack pointer.”**

The address: 0x00007fdc

**Q11: “How many items are on the stack?”**

9 items

**D12: Values on the Stack Screenshot**



**Q13: “Write comments that identify what is stored in each location of the stack.”**

0x00007fdc & 0x00007fe4 & 0x00007fec & 0x00007ff4 & 0x00007ffc: the result of factorial(4) to factorial(0).

0x00007fe0 & 0x00007fe8 & 0x00007ff0 & 0x00007ff8: instructions for loop back to the beginning of the program