

CS322 MINI PROJECT

EXTENDING FUNCTIONALITY OF MARS MIPS ASSEMBLER

REPORT

STACK MEMORY:

Stack is one of the Memory segments in the main memory layout. The role of stack is the storage of temporary data while handling function calls, storage for local variables, passing of parameters in function calls, saving of registers during exception sequences etc.

Stack is mainly used in case of nested subroutine calls, where each of the subroutine may have a set of variables which are local to that subroutine, i.e. their scope is limited only to that function/subroutine call. This temporary data can be conveniently stored on the stack in a stack frame.

STACK FRAME:

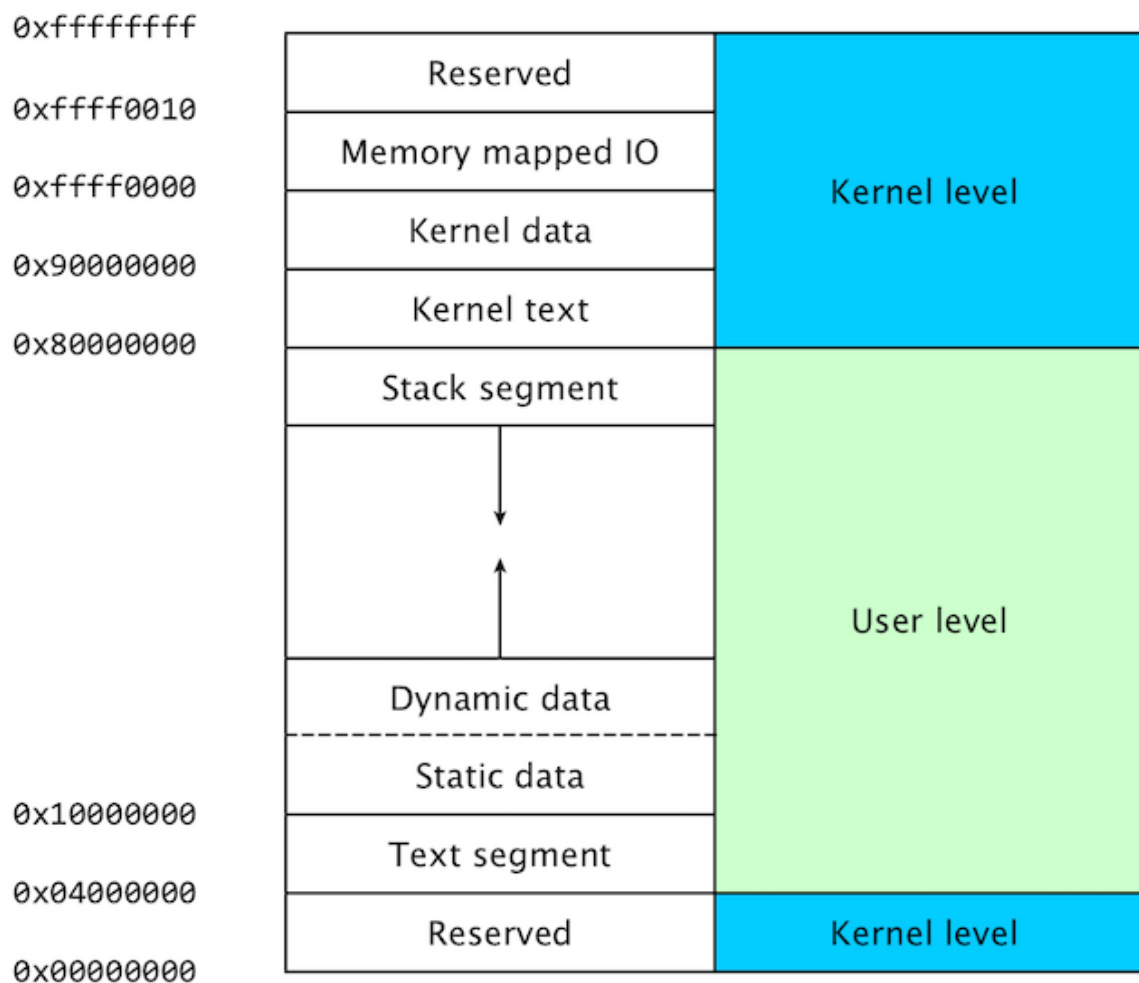
Each function call gets its memory allocated on the stack in the form of a stack frame. Once the function has completed its execution, its stack frame(which contains all the pass arguments and local variables) gets erased from the main memory.

When a function is called, it creates a new frame onto the stack, which will be used for local storage. Before the function returns, it must pop its stack frame, to restore the stack to its original state.

THE MIPS STACK:

In MIPS machines, part of main memory is reserved for a stack. The stack grows downward in terms of memory addresses. — The address of the top element of the stack is stored (by convention) in the “stack pointer” register, \$sp.

MIPS does not provide “push” and “pop” instructions. Instead, they must be done explicitly by the programmer.




MIPS memory layout

MARS STACK VISUALIZER TOOL:

In this project, I have implemented an in-built tool by extending the Abstract Tools class in MARS, which can visually simulate the stack memory in between function calls. It keeps track of:

- Memory address referenced by the stack pointer
- Register being saved in the stack
- Value or address contained in the given register

 Stack Visualizer,
 ✕

Tracking Stack Memory

Address	Word-length Data	Stored Reg	Status
0x7ffffffc	00000000		
0x7ffffff8	00000000	\$s0	
0x7ffffff4	00400008	\$ra	
0x7ffffff0	00000004	\$s0	
0x7fffffec	00400058	\$ra	
0x7fffffe8	00000003	\$s0	
0x7fffffe4	00400058	\$ra	
0x7fffffe0	00000002	\$s0	
0x7fffffdc	00400058	\$ra	
0x7fffffd8	00000001	\$s0	
0x7fffffd4	00000000		
0x7fffffd0	00000000		
0x7fffffcc	00000000		
0x7fffffc8	00000000		
0x7fffffc4	00000000		
0x7fffffc0	00000000		
0x7fffffb8	00000000		
0x7fffffb4	00000000		
0x7fffffb0	00000000		
0x7fffffac	00000000		
0x7fffffa8	00000000		
0x7fffffa4	00000000		
0x7fffffa0	00000000		
0x7fffff9c	00000000		
0x7fffff98	00000000		
0x7fffff94	00000000		
0x7fffff90	00000000		
0x7fffff8c	00000000		
0x7fffff88	00000000		
0x7fffff84	00000000		
0x7fffff80	00000000		
0x7fffff7c	00000000		
0x7fffff78	00000000		
0x7fffff74	00000000		
0x7fffff70	00000000		

Tool Control

Disconnect from MIPS
Reset
Help
Close

The highlighted (yellow) row indicates the current stack pointer location and the greyed rows have not been allocated yet. The rows above the \$sp show the saved register name, value and the address in stack memory.

SIMULATION AND ANALYSIS:

For demonstration purpose, I will simulate the tool using the MIPS codes for:

1. Calculating factorial sequence
2. Calculating fibonacci sequence
3. Calculating fibonacci sequence using Dynamic Programming

The screenshot shows the 'Stack Visualizer' window with the title 'Tracking Stack Memory'. It displays a table of memory addresses, word-length data, stored registers, and status. The first row is highlighted in yellow, showing address 0x7ffefffc with data 00000000. The table lists addresses from 0x7ffefffc down to 0x7ffef70. The 'Word-length Data' column contains 00000000 for all entries. The 'Stored Reg' and 'Status' columns are empty. At the bottom, there is a 'Tool Control' section with four buttons: 'Disconnect from MIPS', 'Reset', 'Help', and 'Close'.

Address	Word-length Data	Stored Reg	Status
0x7ffefffc	00000000		
0x7ffefff8	00000000		
0x7ffefff4	00000000		
0x7ffefff0	00000000		
0x7ffeffec	00000000		
0x7ffeffe8	00000000		
0x7ffeffe4	00000000		
0x7ffeffe0	00000000		
0x7ffefd0c	00000000		
0x7ffefd08	00000000		
0x7ffefd04	00000000		
0x7ffefd00	00000000		
0x7ffefdc0	00000000		
0x7ffefdc8	00000000		
0x7ffefdc4	00000000		
0x7ffefdc0	00000000		
0x7ffefbc0	00000000		
0x7ffefbc8	00000000		
0x7ffefbc4	00000000		
0x7ffefbc0	00000000		
0x7ffefb00	00000000		
0x7ffefb08	00000000		
0x7ffefb04	00000000		
0x7ffefb00	00000000		
0x7ffefa00	00000000		
0x7ffefa08	00000000		
0x7ffefa04	00000000		
0x7ffefa00	00000000		
0x7ffef900	00000000		
0x7ffef908	00000000		
0x7ffef904	00000000		
0x7ffef900	00000000		
0x7ffef800	00000000		
0x7ffef808	00000000		
0x7ffef804	00000000		
0x7ffef800	00000000		
0x7ffef700	00000000		
0x7ffef708	00000000		
0x7ffef704	00000000		
0x7ffef700	00000000		

Tool Control

Disconnect from MIPS Reset Help Close

Initial state of the visualizer

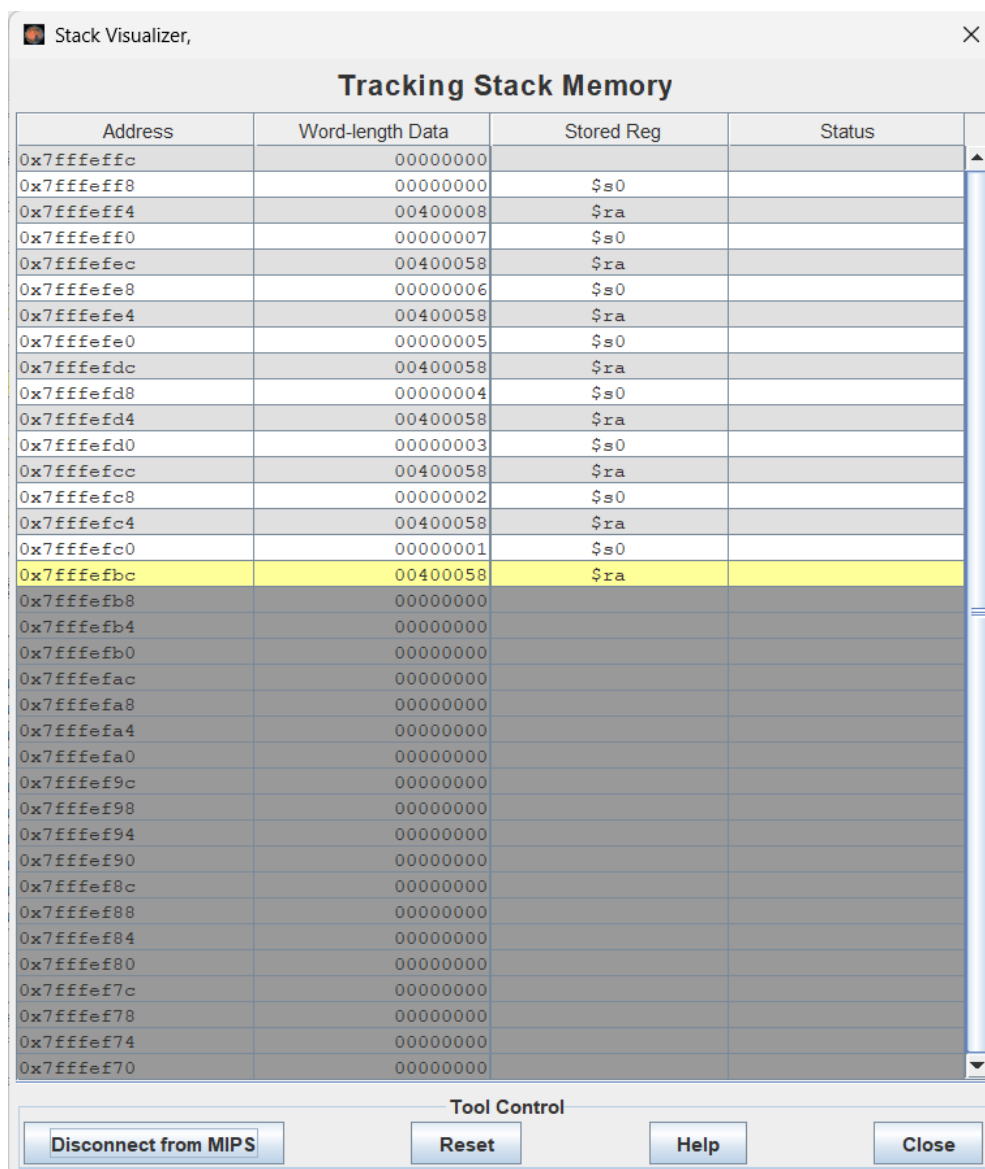
FACTORIAL RECURSIVE CODE:

The process of calculating factorial(N) consists of recursively calling the function alongside reducing the value of N as follows:

$$\text{factorial}(N) = N * \text{factorial}(N-1)$$

where the base case is $N = 0$, which returns the value 1.

Thus, the value of N gets stored on the stack as a function parameter along with the return value in a stack frame. When the stack reaches its max depth, we see that the stack starts unrolling and consequently the frames start getting deleted in a sequence.



Address	Word-length Data	Stored Reg	Status
0x7ffffc	00000000		
0x7ffff8	00000000	\$s0	
0x7ffff4	00400008	\$ra	
0x7ffff0	00000007	\$s0	
0x7fffeec	00400058	\$ra	
0x7fffe8	00000006	\$s0	
0x7fffe4	00400058	\$ra	
0x7fffe0	00000005	\$s0	
0x7fffedc	00400058	\$ra	
0x7ffefd8	00000004	\$s0	
0x7ffefd4	00400058	\$ra	
0x7ffefd0	00000003	\$s0	
0x7ffefcc	00400058	\$ra	
0x7ffefc8	00000002	\$s0	
0x7ffefc4	00400058	\$ra	
0x7ffefc0	00000001	\$s0	
0x7ffefbc	00400058	\$ra	
0x7ffefb8	00000000		
0x7ffefb4	00000000		
0x7ffefb0	00000000		
0x7ffefac	00000000		
0x7ffefa8	00000000		
0x7ffefa4	00000000		
0x7ffefa0	00000000		
0x7ffef9c	00000000		
0x7ffef98	00000000		
0x7ffef94	00000000		
0x7ffef90	00000000		
0x7ffef8c	00000000		
0x7ffef88	00000000		
0x7ffef84	00000000		
0x7ffef80	00000000		
0x7ffef7c	00000000		
0x7ffef78	00000000		
0x7ffef74	00000000		
0x7ffef70	00000000		

Tool Control

Disconnect from MIPS Reset Help Close

At max stack depth (for $N = 7$), we observe 7 stack frames

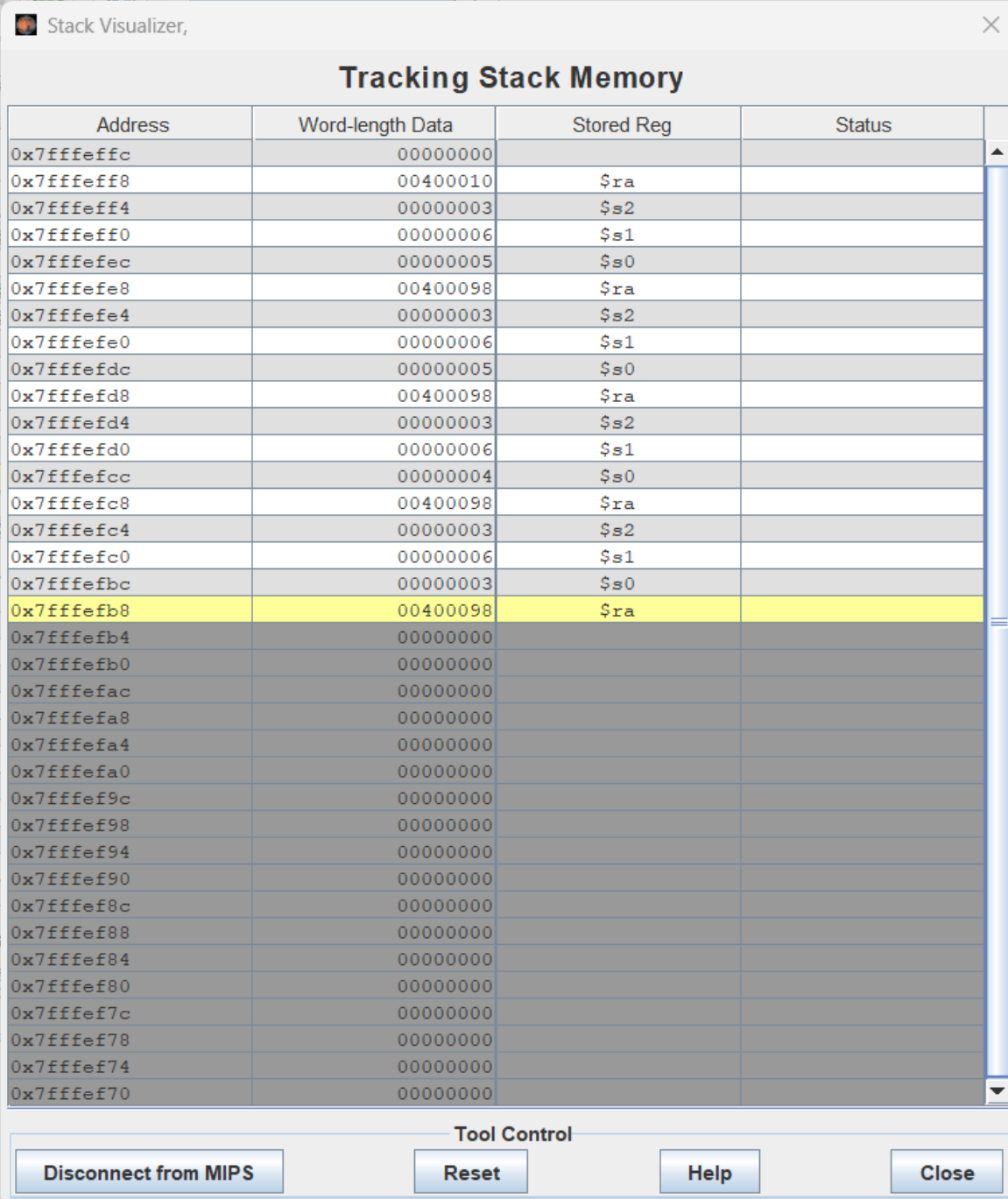
FIBONACCI RECURSIVE CODE:

We recursively calling the function as follows:

$$\text{fib}(N) = \text{fib}(N-1) + \text{fib}(N-2)$$

where the base case is: $\text{fib}(0) = 0$ and $\text{fib}(1) = 1$

Thus, we store the value belonging to one function call in one frame stack which gets deleted after the unwinding of stack starts.



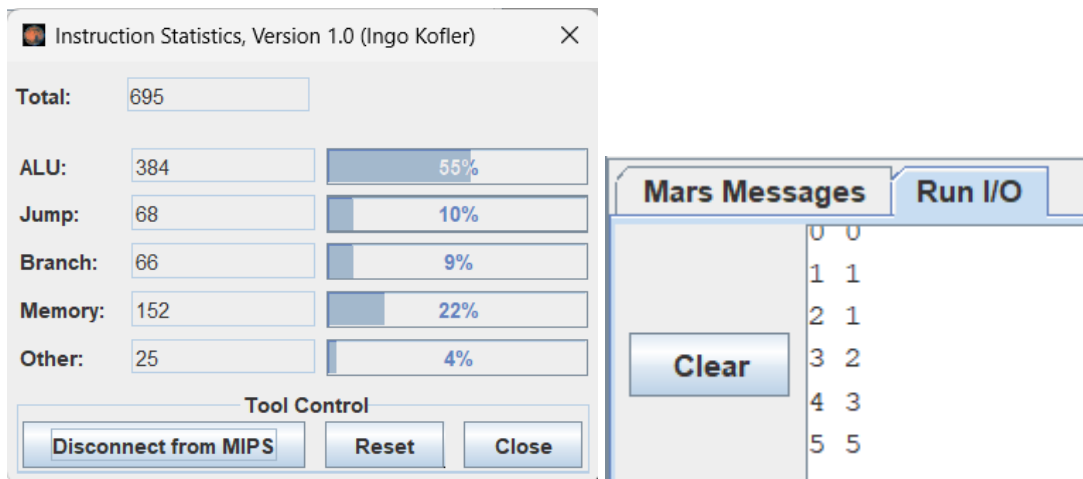
The image shows a 'Stack Visualizer' window titled 'Tracking Stack Memory'. It displays a table of stack memory addresses, word-length data, stored registers, and status. The table is scrollable, and the entry at address 0x7fffefb8 is highlighted in yellow. The registers \$ra, \$s2, and \$s1 are shown in the 'Stored Reg' column. The 'Tool Control' bar at the bottom contains buttons for 'Disconnect from MIPS', 'Reset', 'Help', and 'Close'.

Address	Word-length Data	Stored Reg	Status
0x7fffeffc	00000000		
0x7fffeff8	00400010	\$ra	
0x7fffeff4	00000003	\$s2	
0x7fffeff0	00000006	\$s1	
0x7fffefec	00000005	\$s0	
0x7fffef8	00400098	\$ra	
0x7fffef4	00000003	\$s2	
0x7fffef0	00000006	\$s1	
0x7fffedc	00000005	\$s0	
0x7fffed8	00400098	\$ra	
0x7fffed4	00000003	\$s2	
0x7fffed0	00000006	\$s1	
0x7fffec	00000004	\$s0	
0x7fffec8	00400098	\$ra	
0x7fffec4	00000003	\$s2	
0x7fffec0	00000006	\$s1	
0x7fffeb8	00000003	\$s0	
0x7fffeb8	00400098	\$ra	
0x7fffeb4	00000000		
0x7fffeb0	00000000		
0x7ffefac	00000000		
0x7ffefa8	00000000		
0x7ffefa4	00000000		
0x7ffefa0	00000000		
0x7ffef9c	00000000		
0x7ffef98	00000000		
0x7ffef94	00000000		
0x7ffef90	00000000		
0x7ffef8c	00000000		
0x7ffef88	00000000		
0x7ffef84	00000000		
0x7ffef80	00000000		
0x7ffef7c	00000000		
0x7ffef78	00000000		
0x7ffef74	00000000		
0x7ffef70	00000000		

Tool Control

Disconnect from MIPS Reset Help Close

At max depth (for N = 5)

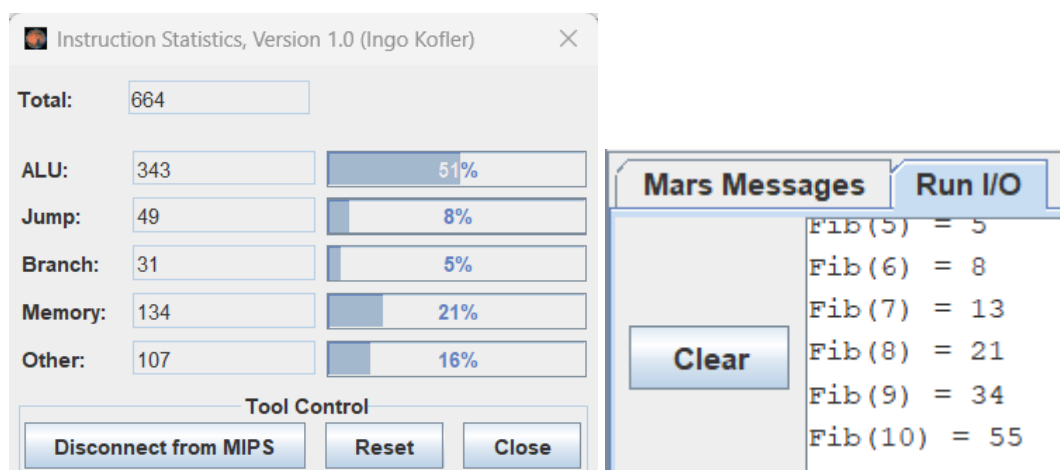


(N = 5)

FIBONACCI RECURSIVE CODE WITH DYNAMIC PROGRAMMING:

The procedure remains the same as in general fibonacci sequence calculation, except we first check if a particular value has been calculated and stored in the data memory before. We call (and thus allocate a stack frame) only if the function has not been called with that argument yet.

This makes our program much more efficient in terms of time complexity as well as the stack memory used.



(N = 10)

Stack Visualizer, ×

Tracking Stack Memory

Address	Word-length Data	Stored Reg	Status
0x7ffffffc	00000000		
0x7ffffff8	00400008	\$ra	
0x7ffffff4	0000000a	\$a0	
0x7ffffff0	004000ac	\$ra	
0x7fffffec	00000009	\$a0	
0x7fffffe8	004000ac	\$ra	
0x7fffffe4	00000008	\$a0	
0x7fffffe0	004000ac	\$ra	
0x7fffffdc	00000007	\$a0	
0x7fffffd8	004000ac	\$ra	
0x7fffffd4	00000006	\$a0	
0x7fffffd0	004000ac	\$ra	
0x7fffffcc	00000005	\$a0	
0x7fffffc8	004000ac	\$ra	
0x7fffffc4	00000004	\$a0	
0x7fffffc0	004000ac	\$ra	
0x7fffffb8	00000003	\$a0	
0x7fffffb4	004000ac	\$ra	
0x7fffffb0	00000002	\$a0	
0x7fffffac	004000ac	\$ra	
0x7fffffa8	00000001	\$v0	
0x7fffffa4	00000000		
0x7fffffa0	00000000		
0x7ffff9fc	00000000		
0x7ffff9f8	00000000		
0x7ffff9f4	00000000		
0x7ffff9f0	00000000		
0x7ffff98c	00000000		
0x7ffff988	00000000		
0x7ffff984	00000000		
0x7ffff980	00000000		
0x7ffff97c	00000000		
0x7ffff978	00000000		
0x7ffff974	00000000		
0x7ffff970	00000000		

Tool Control

At max depth (For N = 10)

It can be noted that dynamic programming makes the program much more efficient since it does not need to call the same routine multiple times redundantly if the return value for that has been stored in a data memory location retrieved from the stack before the frame can be erased.

SUMMARY:

I have created a Stack Visualizer tool which keeps track of the stack pointer and visually simulates the stack memory in between nested function calls, storing local variables, caller and callee-saved registers, function parameters and arguments on the stack, in the form of stack frames which are erased at the end of function execution.

SUBMITTED BY: Animesh Kumar Sinha
ROLL NUMBER: 2001CS07