

Stack Processor

Generated by Doxygen 1.8.10

Fri Oct 23 2015 13:40:29

Contents

1	Todo List	1
2	Data Structure Index	3
2.1	Data Structures	3
3	File Index	5
3.1	File List	5
4	Data Structure Documentation	7
4.1	CPU Struct Reference	7
4.1.1	Detailed Description	7
4.1.2	Field Documentation	7
4.1.2.1	rax	7
4.1.2.2	stack	7
4.1.2.3	state	7
4.2	Stack Struct Reference	8
4.2.1	Detailed Description	8
4.2.2	Field Documentation	8
4.2.2.1	state	8
4.2.2.2	top	8
4.2.2.3	values	8
5	File Documentation	9
5.1	CPU.h File Reference	9
5.1.1	Function Documentation	10
5.1.1.1	CPU_add(CPU *This)	10
5.1.1.2	CPU_construct(CPU *This)	10
5.1.1.3	CPU_construct_copy(CPU *This, const CPU *other)	10
5.1.1.4	CPU_destruct(CPU *This)	10
5.1.1.5	CPU_div(CPU *This)	11
5.1.1.6	CPU_dump_(const CPU *This, const char name[])	11
5.1.1.7	CPU_mul(CPU *This)	11
5.1.1.8	CPU_OK(const CPU *This)	11

5.1.1.9	CPU_pop(CPU *This, TYPE *register_)	12
5.1.1.10	CPU_pow(CPU *This)	12
5.1.1.11	CPU_push(CPU *This, const TYPE *register_)	12
5.1.1.12	CPU_sub(CPU *This)	13
5.2	stack.h File Reference	13
5.2.1	Function Documentation	14
5.2.1.1	stack_construct(Stack *This)	14
5.2.1.2	stack_construct_copy(Stack *This, const Stack *other)	14
5.2.1.3	stack_destruct(Stack *This)	14
5.2.1.4	stack_dump_(const Stack *This, const char name[])	15
5.2.1.5	Stack_OK(const Stack *This)	15
5.2.1.6	stack_pop(Stack *This)	15
5.2.1.7	stack_push(Stack *This, TYPE value)	15
Index		17

Chapter 1

Todo List

globalScope> Global CPU_construct (CPU *This)

Dynamic memory management, so its OK that only true is returned by now.

globalScope> Global stack_construct (Stack *This)

Dynamic memory management, so its OK that only true is returned by now

Chapter 2

Data Structure Index

2.1 Data Structures

Here are the data structures with brief descriptions:

CPU	Simple stack processor	7
Stack	Simple stack of integer	8

Chapter 3

File Index

3.1 File List

Here is a list of all documented files with brief descriptions:

CPU.h	9
more_errors.h	??
mylib.h	??
stack.h	13

Chapter 4

Data Structure Documentation

4.1 CPU Struct Reference

Simple stack processor.

```
#include <CPU.h>
```

Data Fields

- [TYPE rax](#)
- [Stack stack](#)
- bool [state](#)

4.1.1 Detailed Description

Simple stack processor.

Processor is a structure that contains a stack and a register to operate with.

4.1.2 Field Documentation

4.1.2.1 TYPE CPU::rax

Register (64-bit).

4.1.2.2 Stack CPU::stack

Operating stack.

4.1.2.3 bool CPU::state

State of the [CPU](#). true if ON, false if OFF.

The documentation for this struct was generated from the following file:

- [CPU.h](#)

4.2 Stack Struct Reference

Simple stack of integer.

```
#include <stack.h>
```

Data Fields

- [TYPE values \[MAX_SIZE\]](#)
- [TYPE * top](#)
- [bool state](#)

4.2.1 Detailed Description

Simple stack of integer.

[Stack](#) contains integer numbers and provides some operations with them

4.2.2 Field Documentation

4.2.2.1 `bool Stack::state`

State of the stack. true if valid, false otherwise.

4.2.2.2 `TYPE* Stack::top`

Pointer to the top element of stack.

4.2.2.3 `TYPE Stack::values[MAX_SIZE]`

Array of values.

The documentation for this struct was generated from the following file:

- [stack.h](#)

Chapter 5

File Documentation

5.1 CPU.h File Reference

```
#include "CPU.h"  
#include <assert.h>  
#include <errno.h>
```

Data Structures

- struct [CPU](#)
Simple stack processor.

Functions

- bool [CPU_construct](#) ([CPU](#) *This)
Standard CPU constructor.
- bool [CPU_construct_copy](#) ([CPU](#) *This, const [CPU](#) *other)
Copy CPU constructor.
- void [CPU_destruct](#) ([CPU](#) *This)
Destructs the CPU.
- bool [CPU_OK](#) (const [CPU](#) *This)
Validates the CPU.
- void [CPU_dump_](#) (const [CPU](#) *This, const char name[])
Prints CPU's dump.
- bool [CPU_push](#) ([CPU](#) *This, const [TYPE](#) *register_)
Pushes value to CPU stack.
- bool [CPU_pop](#) ([CPU](#) *This, [TYPE](#) *register_)
Pops value from the CPU.
- bool [CPU_add](#) ([CPU](#) *This)
Summs the top two elements of the stack.
- bool [CPU_sub](#) ([CPU](#) *This)
Subtracts the penult stack element from the top one.
- bool [CPU_mul](#) ([CPU](#) *This)
Multiplies the top two elements of the stack.
- bool [CPU_div](#) ([CPU](#) *This)
Divides the top stack element by the previous.
- bool [CPU_pow](#) ([CPU](#) *This)
Raise the top element in the power of the penult one.

5.1.1 Function Documentation

5.1.1.1 `bool CPU_add (CPU * This)`

Summs the top two elements of the stack.

Add the top element of stack to the previous and stroes the result in stack. Both top two elements are removed.

Parameters

<i>This</i>	Pointer to the <code>CPU</code> to perform operation on.
-------------	--

Returns

true if success, false otherwise. In case of fail invalidates `CPU`.

Warning

`Stack` must contain at least two elements.

5.1.1.2 `bool CPU_construct (CPU * This)`

Standard `CPU` constructor.

Constructs `CPU` with empty `CPU` and register.

Parameters

<i>This</i>	Pointer to the <code>CPU</code> to be constructed.
-------------	--

Returns

1 (true) if success, 0 (false) otherwise.

Todo Dynamic memory management, so its OK that only true is returned by now.

5.1.1.3 `bool CPU_construct_copy (CPU * This, const CPU * other)`

Copy `CPU` constructor.

Constructs `CPU` as copy of other. Writes to errno.

Parameters

<i>This</i>	Pointer to the <code>CPU</code> to be constructed.
<i>other</i>	The <code>CPU</code> to copy from.

Returns

1 (true) if success, 0 (false) otherwise.

5.1.1.4 `void CPU_destruct (CPU * This)`

Destructs the `CPU`.

Destructs the `CPU`, setting its values to poison.

Parameters

<i>This</i>	Pointer to the CPU to be destructed.
-------------	--

5.1.1.5 `bool CPU_div (CPU * This)`

Divides the top stack element by the previous.

Gets the top element of stack and divides it by the penult stack element. Both top two elements are removed and the result is written to stack.

Parameters

<i>This</i>	Pointer to the CPU to perform operation on.
-------------	---

Returns

true if success, false otherwise. In case of fail invalidates [CPU](#).

Warning

[Stack](#) must contain at least two elements.

5.1.1.6 `void CPU_dump_ (const CPU * This, const char name[])`

Prints [CPU](#)'s dump.

Outputs the current state of [CPU](#).

Parameters

<i>This</i>	Pointer to the CPU to be dumped.
-------------	--

5.1.1.7 `bool CPU_mul (CPU * This)`

Multiplies the top two elements of the stack.

Multiplies the top element of stack to the previous and stroes the result in stack. Both top two elements are removed.

Parameters

<i>This</i>	Pointer to the CPU to perform operation on.
-------------	---

Returns

true if success, false otherwise. In case of fail invalidates [CPU](#).

Warning

[Stack](#) must contain at least two elements.

5.1.1.8 `bool CPU_OK (const CPU * This)`

Validates the [CPU](#).

Checks if the [CPU](#) is correct according to its values.

Parameters

<i>This</i>	Pointer to the CPU to be checked.
-------------	---

Returns

true if the [CPU](#) is valid, false otherwise.

5.1.1.9 bool CPU_pop (CPU * *This*, TYPE * *register_*)

Pops value from the [CPU](#).

Pops the top element from the [CPU](#) and returns it.

Parameters

<i>This</i>	Pointer to the CPU to perform operation on.
<i>register_</i>	Register where popped value is stored.

Returns

true if success, false otherwise. In case it wasn't successful, invalidates [CPU](#).

Warning

[Stack](#) must contain at least one element.

5.1.1.10 bool CPU_pow (CPU * *This*)

Raise the top element in the power of the penult one.

Gets the top element of stack and raises it to power equal to the penult stack element. Both top two elements are removed and the result is written to stack.

Parameters

<i>This</i>	Pointer to the CPU to perform operation on.
-------------	---

Returns

true if success, false otherwise. In case of fail invalidates [CPU](#).

Warning

[Stack](#) must contain at least two elements.

5.1.1.11 bool CPU_push (CPU * *This*, const TYPE * *register_*)

Pushes value to [CPU](#) stack.

Puts the value from given register at the top of [CPU](#) stack.

Parameters

<i>This</i>	Pointer to the CPU to perform operation on.
<i>register_</i>	Address of the register where value is stored.

Returns

true if success, false otherwise.

5.1.1.12 bool CPU_sub (CPU * This)

Subtracts the penult stack element from the top one.

Gets the top element of stack and subtracts the penult stack element from it. Both top two elements are removed and the result is written to stack.

Parameters

<i>This</i>	Pointer to the CPU to perform operation on.
-------------	---

Returns

true if success, false otherwise. In case of fail invalidates [CPU](#).

Warning

[Stack](#) must contain at least two elements.

5.2 stack.h File Reference

```
#include <assert.h>
#include <errno.h>
#include <string.h>
```

Data Structures

- struct [Stack](#)
Simple stack of integer.

Macros

- #define [TYPE](#) int
Type of stack's values.
- #define [MAX_SIZE](#) 16
Maximum number of elements in stack.
- #define [MAX_PRINTED](#) [MAX_SIZE](#)
- #define [Stack_dump](#)(This) [stack_dump_](#)(This, #This)
More comfortable dump.
- #define [stack_dump](#)(This) [Stack_dump](#)(This)
To be stylish.
- #define [Stack_OK](#)(This) [stack_OK](#)(This)

Functions

- bool `stack_construct` (`Stack *This`)
Standard stack constructor.
- bool `stack_construct_copy` (`Stack *This`, const `Stack *other`)
Copy stack constructor.
- void `stack_destruct` (`Stack *This`)
Destructs the stack.
- bool `Stack_OK` (const `Stack *This`)
Validates the stack.
- void `stack_dump_` (const `Stack *This`, const char name[])
Prints stack's dump.
- bool `stack_push` (`Stack *This`, `TYPE` value)
Pushes value to stack.
- `TYPE` `stack_pop` (`Stack *This`)
Pops value from the stack.

5.2.1 Function Documentation

5.2.1.1 bool stack_construct (Stack * This)

Standard stack constructor.

Constructs stack with the top pointer at NULL. Writes to errno.

Parameters

<i>This</i>	Pointer to the stack to be constructed.
-------------	---

Returns

1 (true) if success, 0 (false) otherwise.

Todo Dynamic memory management, so its OK that only true is returned by now

5.2.1.2 bool stack_construct_copy (Stack * This, const Stack * other)

Copy stack constructor.

Constructs stack as copy of other. Writes to errno.

Parameters

<i>This</i>	Pointer to the stack to be constructed.
<i>other</i>	The stack to copy from.

Returns

1 (true) if success, 0 (false) otherwise.

5.2.1.3 void stack_destruct (Stack * This)

Destructs the stack.

Destructs the stack, setting its values to poison.

Parameters

<i>This</i>	Pointer to the stack to be destructed.
-------------	--

5.2.1.4 void stack_dump_ (const Stack * *This*, const char *name*[])

Prints stack's dump.

Outputs the current state of stack.

Parameters

<i>This</i>	Pointer to the stack to be dumped.
-------------	------------------------------------

5.2.1.5 bool Stack_OK (const Stack * *This*)

Validates the stack.

Checks if the stack is correct according to its values.

Parameters

<i>This</i>	Pointer to the stack to be checked.
-------------	-------------------------------------

Returns

true if the stack is valid, false otherwise.

5.2.1.6 TYPE stack_pop (Stack * *This*)

Pops value from the stack.

Pops the top element from the stack and returns it.

Parameters

<i>This</i>	Pointer to the stack to perform operation on.
-------------	---

Returns

Value that was popped. In case it wasn't successful, invalidates stack.

5.2.1.7 bool stack_push (Stack * *This*, TYPE *value*)

Pushes value to stack.

Put the given value to stack as top element.

Parameters

<i>This</i>	Pointer to the stack to perform operation on.
<i>value</i>	Value to be pushed.

Returns

true if success, false otherwise

Index

CPU, [7](#)

 rax, [7](#)

 stack, [7](#)

 state, [7](#)

CPU.h, [9](#)

 CPU_OK, [11](#)

 CPU_add, [10](#)

 CPU_construct, [10](#)

 CPU_construct_copy, [10](#)

 CPU_destruct, [10](#)

 CPU_div, [11](#)

 CPU_dump_, [11](#)

 CPU_mul, [11](#)

 CPU_pop, [12](#)

 CPU_pow, [12](#)

 CPU_push, [12](#)

 CPU_sub, [13](#)

CPU_OK

 CPU.h, [11](#)

CPU_add

 CPU.h, [10](#)

CPU_construct

 CPU.h, [10](#)

CPU_construct_copy

 CPU.h, [10](#)

CPU_destruct

 CPU.h, [10](#)

CPU_div

 CPU.h, [11](#)

CPU_dump_

 CPU.h, [11](#)

CPU_mul

 CPU.h, [11](#)

CPU_pop

 CPU.h, [12](#)

CPU_pow

 CPU.h, [12](#)

CPU_push

 CPU.h, [12](#)

CPU_sub

 CPU.h, [13](#)

rax

 CPU, [7](#)

Stack, [8](#)

 state, [8](#)

 top, [8](#)

 values, [8](#)

stack

CPU, [7](#)

stack.h, [13](#)

 Stack_OK, [15](#)

 stack_construct, [14](#)

 stack_construct_copy, [14](#)

 stack_destruct, [14](#)

 stack_dump_, [15](#)

 stack_pop, [15](#)

 stack_push, [15](#)

Stack_OK

 stack.h, [15](#)

stack_construct

 stack.h, [14](#)

stack_construct_copy

 stack.h, [14](#)

stack_destruct

 stack.h, [14](#)

stack_dump_

 stack.h, [15](#)

stack_pop

 stack.h, [15](#)

stack_push

 stack.h, [15](#)

state

 CPU, [7](#)

 Stack, [8](#)

top

 Stack, [8](#)

values

 Stack, [8](#)