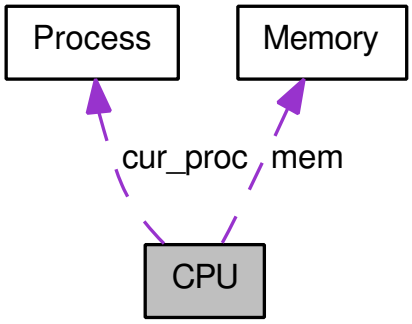
4.2 CPU Class Reference

Represents the CPU of the simulated machine.

`#include <CPU.h>`Collaboration diagram for CPU:



Public Member Functions

- CPU (Memory *m=NULL, Process *p=NULL)
- virtual ~CPU ()
- int execute ()

  Executes a single instruciton.

- void load_process (Process *p)

  Load the current process.

Private Member Functions

- int pop ()
- void push (int)
- void print_stack ()
- void print_register ()
- const char * instr_to_string (int instr)
- int tlb_have_entry (mem_addr_t virtual_addr)

  Does the TLB have an entry for the virtual address.

- mem_addr_t tlb_translate (mem_addr_t virtual_addr)

  Translate a virtual address to a physical address.

- int ld_instr ()
  LD CPU Instruction.
- int la_instr ()
  LA CPU Instruction.
- int load_instr ()
  LOAD CPU Instruction.
- int loadi_instr ()
  LOADI CPU Instruction.
- int add_instr ()
  ADD CPU Instruction.
- int sub_instr ()
  SUB CPU Instruction.
- int mul_instr ()
  MUL CPU Instruction.
- int div_instr ()
  DIV CPU Instruction.
- int end_instr ()
  END CPU Instruction.
- int endp_instr ()
  ENDP CPU Instruction.
- int and_instr ()
  AND CPU Instruction.
- int or_instr ()
  OR CPU Instruction.
- int not_instr ()
  NOT CPU Instruction.
- int le_op_instr ()
  LE_OP CPU Instruction.

- int ge_op_instr ()

  GE_OP CPU Instruction.

- int lt_op_instr ()

  LT_OP CPU Instruction.

- int gt_op_instr ()

  GT_OP CPU Instruction.

- int eq_op_instr ()

  EQ_OP CPU Instruction.

- int ne_op_instr ()

  NE_OP CPU Instruction.

- int stop_instr ()

  STOP CPU Instruction.

- int stor_instr ()

  STOR CPU Instruction.

- int st_instr ()

  ST CPU Instruction.

- int lock_instr ()

  LOCK CPU Instruction.

- int unlock_instr ()

  UNLOCK CPU Instruction.

- int halt_instr ()

  HALT CPU Instruction.

- int jfalse_instr ()

  JFALSE CPU Instruction.

- int jmp_instr ()

  JMP CPU Instruction.

Private Attributes

- int reg [REGISTERSIZE]
- int stack [STACKSIZE]
- Memory * mem

  Memory attached to this CPU.

- mem_addr_t ip

  Instruction pointer.

- int sp

  Stack pointer: position within current process stack.

- int offset

  To support memory relocation in processes until VMM is enabled.

- Process * cur_proc

  Current process.

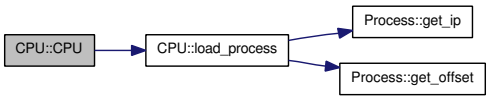
4.2.1 Detailed Description

Represents the CPU of the simulated machine. The CPU takes instructions from the attached memory and executes one instruction each time execute() is called.

4.2.2 Constructor & Destructor Documentation

4.2.2.1 CPU::CPU (Memory * *m* = `NULL`, Process * *p* = `NULL`)

Here is the call graph for this function:



4.2.2.2 CPU::~CPU () [virtual]

4.2.3 Member Function Documentation

4.2.3.1 int CPU::add_instr () [private]

ADD CPU Instruction. Adds two top values on stack

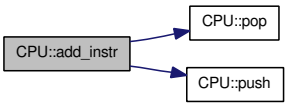
1. Pops 1st value from stack
2. Pops 2nd value from stack
3. Adds values
4. Pushes answer back onto stack

Instruction length is 1 int [instr]

Returns:

CPU return status such as interupts or OS calls

Here is the call graph for this function:



Here is the caller graph for this function:

4.2.3.2 int CPU::and_instr () [private]

AND CPU Instruction. Logically ANDs two top values on stack

1. Pops 1st value from stack
2. Pops 2nd value from stack
3. ANDs int values (&&)

4. Pushes answer back onto stack

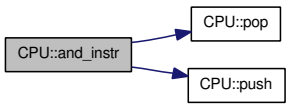
Instruction length is 1 int [instr]

TODO: Should be bitwise?

Returns:

CPU return status such as interupts or OS calls

Here is the call graph for this function:



Here is the caller graph for this function:

4.2.3.3 int CPU::div_instr () [private]

DIV CPU Instruction. Divides top value on stack by the 2nd to top value

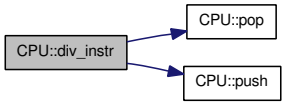
1. Pops 1st value from stack
2. Pops 2nd value from stack
3. Divides 1st by 2nd value
4. Pushes answer back onto stack

Instruction length is 1 int [instr]

Returns:

CPU return status such as interupts or OS calls

Here is the call graph for this function:



Here is the caller graph for this function:

4.2.3.4 int CPU::end_instr () [private]

END CPU Instruction. End of process isntruction. Process should be removed.

Instruction length is 1 int [instr]

Returns:

CPU return status such as interupts or OS calls

Here is the caller graph for this function:

4.2.3.5 int CPU::endp_instr () [private]

ENDP CPU Instruction. This instruction is not implemented

Returns:

CPU return status such as interupts or OS calls

Here is the caller graph for this function:

4.2.3.6 int CPU::eq_op_instr () [private]

EQ_OP CPU Instruction. Determines if the top value on the stack is equal to the 2nd to top value.

1. Pops 1st value from stack

2. Pops 2nd value from stack

3. Logically compares to see if 1st value is equal to the second value

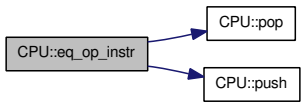
4. Pushes answer back onto stack

Instruction length is 1 int [instr]

Returns:

CPU return status such as interupts or OS calls

Here is the call graph for this function:



Here is the caller graph for this function:

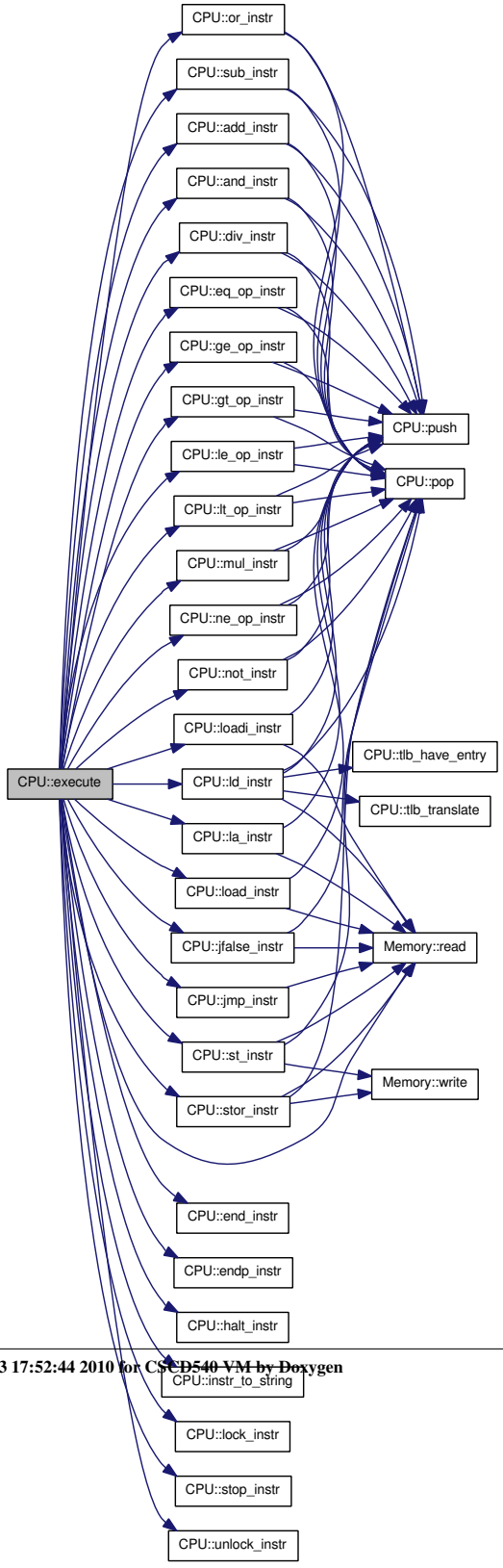
4.2.3.7 int CPU::execute ()

Executes a single instruciton. Based on the current process, this method executes a single instruction. The returned CPU result flag will indicate whether or not the CPU succeeded in executing a command or interupt values. See class constants.

Returns:

CPU result flag

Here is the call graph for this function:



4.2.3.8 int CPU::ge_op_instr () [private]

GE_OP CPU Instruction. Determines if the top value on the stack is greater than or equal to the 2nd to top value.

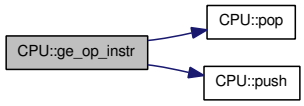
1. Pops 1st value from stack
2. Pops 2nd value from stack
3. Logically compares to see if 1st value is greater than or equal to second value
4. Pushes answer back onto stack

Instruction length is 1 int [instr]

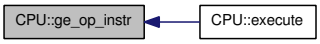
Returns:

CPU return status such as interupts or OS calls

Here is the call graph for this function:



Here is the caller graph for this function:



4.2.3.9 int CPU::gt_op_instr () [private]

GT_OP CPU Instruction. Determines if the top value on the stack is greater than the 2nd to top value.

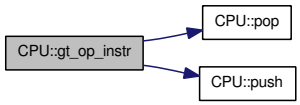
1. Pops 1st value from stack
2. Pops 2nd value from stack
3. Logically compares to see if 1st value is greater than the second value
4. Pushes answer back onto stack

Instruction length is 1 int [instr]

Returns:

CPU return status such as interupts or OS calls

Here is the call graph for this function:



Here is the caller graph for this function:

4.2.3.10 int CPU::halt_instr () [private]

HALT CPU Instruction. Performs a premature halting of the running application (all processes).

Instruction length is 1 int [instr]

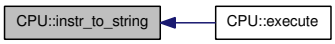
Returns:

CPU return status such as interupts or OS calls

Here is the caller graph for this function:

4.2.3.11 const char * CPU::instr_to_string (int *instr*) [private]

Here is the caller graph for this function:



4.2.3.12 int CPU::jfalse_instr () [private]

JFALSE CPU Instruction. Jumps to immediate address if top value on stack is logically false (0)

1. Pops value from stack
2. Reads immediate virtual address from instruction memory
3. If value is false
    - Jump to immediate address (see below)

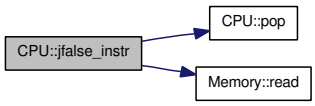
There's some interesting calculations going on here. See code for modifications to immediate address target.

Instruction length is 2 ints [instr][address]

Returns:

CPU return status such as interupts or OS calls

Here is the call graph for this function:



Here is the caller graph for this function:

4.2.3.13 int CPU::jmp_instr () [private]

JMP CPU Instruction. Jumps to immediate address

1. Reads immediate virtual address from instruction memory
2. Jumps to immediate address (see below)

There's some interesting calculations going on here. See code for modifications to immediate address target.

Instruction length is 2 ints [instr][address]

Returns:

CPU return status such as interupts or OS calls

Here is the call graph for this function:

Here is the caller graph for this function:

4.2.3.14 int CPU::la_instr () [private]

LA CPU Instruction. Load immediate address to stack

1. Reads immediate address from instruction memory
2. Pushes immediate address to stack

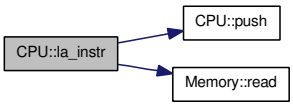
Typically used to load the start of an array to stack

Instruction length is 2 ints [instr][address]

Returns:

CPU return status such as interupts or OS calls

Here is the call graph for this function:



Here is the caller graph for this function:

4.2.3.15 int CPU::ld_instr () [private]

LD CPU Instruction. Load value from memory

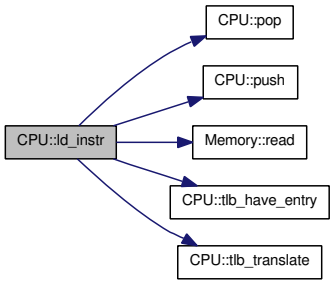
1. Pops a (virtual) memory address from the stack
2. Translates virtual address to physical
3. Reads int from physical memory
4. Pushes read value to

Instruction length is 1 int [instr]

Returns:

CPU return status such as interupts or OS calls

Here is the call graph for this function:



Here is the caller graph for this function:

4.2.3.16 int CPU::le_op_instr () [private]

LE_OP CPU Instruction. Determines if the top value on the stack is less than or equal to the 2nd to top value.

1. Pops 1st value from stack
2. Pops 2nd value from stack

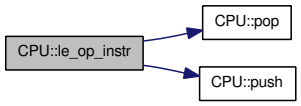
3. Logically compares to see if 1st value is less than or equal to second value
4. Pushes answer back onto stack

Instruction length is 1 int [instr]

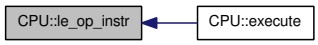
Returns:

CPU return status such as interupts or OS calls

Here is the call graph for this function:



Here is the caller graph for this function:



4.2.3.17 int CPU::load_instr () [private]

LOAD CPU Instruction. Reads a value from memory and pushes it to the stack.

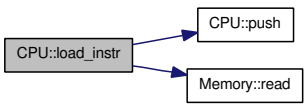
1. Reads immediate virtual address from instruction memory
2. Reads value from memory
3. Pushes value to stack

Instruction length is 2 ints [instr][address]

Returns:

CPU return status such as interupts or OS calls

Here is the call graph for this function:



Here is the caller graph for this function:

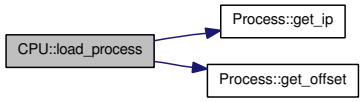
4.2.3.18 void CPU::load_process (Process * *p*)

Load the current process.

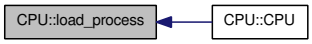
Returns:

CPU result flag

Here is the call graph for this function:



Here is the caller graph for this function:



4.2.3.19 int CPU::loadi_instr () `[private]`

LOADI CPU Instruction. Push immediate value to the stack

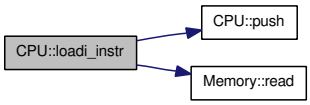
1. Reads immediate value from instruction memory
2. Pushes value to stack

Instruction length is 2 ints [instr][value]

Returns:

CPU return status such as interupts or OS calls

Here is the call graph for this function:



Here is the caller graph for this function:

4.2.3.20 int CPU::lock_instr () [private]

LOCK CPU Instruction. Gives this process exclusive access to machine preventing any other processes from running.

Instruction length is 1 int [instr]

Returns:

CPU return status such as interupts or OS calls

Here is the caller graph for this function:

4.2.3.21 int CPU::lt_op_instr () [private]

LT_OP CPU Instruction. Determines if the top value on the stack is less than the 2nd to top value.

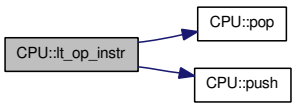
1. Pops 1st value from stack
2. Pops 2nd value from stack
3. Logically compares to see if 1st value is less than the second value
4. Pushes answer back onto stack

Instruction length is 1 int [instr]

Returns:

CPU return status such as interupts or OS calls

Here is the call graph for this function:



Here is the caller graph for this function:

4.2.3.22 int CPU::mul_instr () [private]

MUL CPU Instruction. Multiplies two top values on stack

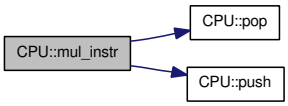
1. Pops 1st value from stack
2. Pops 2nd value from stack
3. Multiples values
4. Pushes answer back onto stack

Instruction length is 1 int [instr]

Returns:

CPU return status such as interupts or OS calls

Here is the call graph for this function:



Here is the caller graph for this function:

4.2.3.23 int CPU::ne_op_instr () [private]

NE_OP CPU Instruction. Determines if the top value on the stack is not equal to the 2nd to top value.

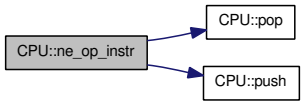
1. Pops 1st value from stack
2. Pops 2nd value from stack
3. Logically compares to see if 1st value is not equal to the second value
4. Pushes answer back onto stack

Instruction length is 1 int [instr]

Returns:

CPU return status such as interupts or OS calls

Here is the call graph for this function:



Here is the caller graph for this function:

4.2.3.24 int CPU::not_instr () [private]

NOT CPU Instruction. Negates top value on stack

1. Pops value from stack
2. Logically negates int value (!)
3. Pushes answer back onto stack

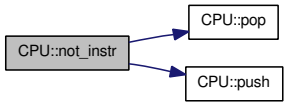
Instruction length is 1 int [instr]

TODO: Should be bitwise?

Returns:

CPU return status such as interupts or OS calls

Here is the call graph for this function:



Here is the caller graph for this function:

4.2.3.25 int CPU::or_instr () [private]

OR CPU Instruction. Logically ORs two top values on stack

1. Pops 1st value from stack
2. Pops 2nd value from stack
3. ORs int values (&&)
4. Pushes answer back onto stack

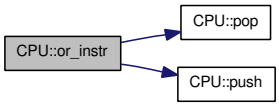
Instruction length is 1 int [instr]

TODO: Should be bitwise?

Returns:

CPU return status such as interupts or OS calls

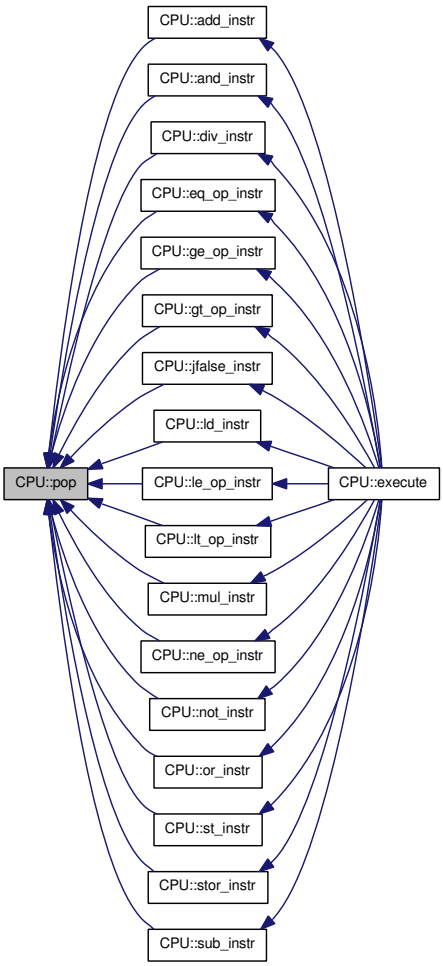
Here is the call graph for this function:



Here is the caller graph for this function:

4.2.3.26 int CPU::pop () [private]

Here is the caller graph for this function:

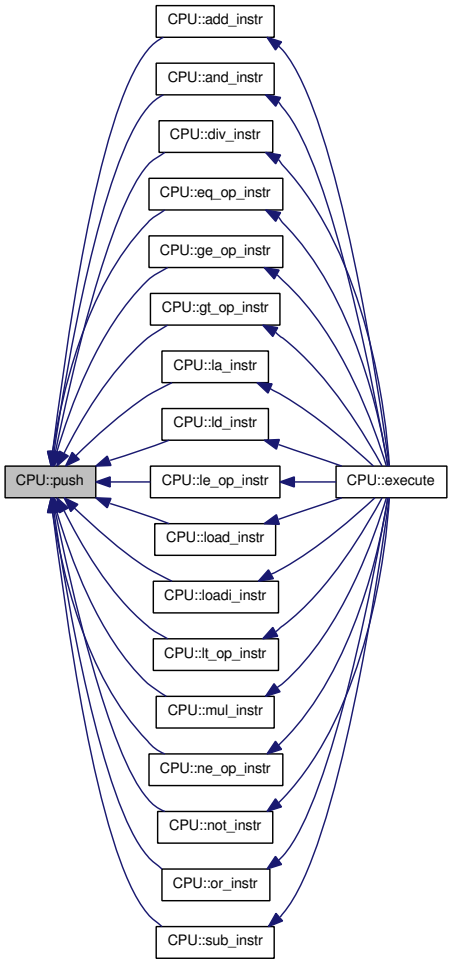


4.2.3.27 void CPU::print_register () [private]

4.2.3.28 void CPU::print_stack () [private]

4.2.3.29 void CPU::push (int *data*) [private]

Here is the caller graph for this function:



4.2.3.30 int CPU::st_instr () [private]

ST CPU Instruction. Stores a value at the 2nd from top position on the stack to the address in memory stored as the top value on the stack.

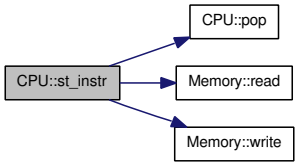
1. Pops virtual address from the stack
2. Pops value to store
3. Write value to memory at address from stack

Instruction length is 1 int [instr]

Returns:

CPU return status such as interupts or OS calls

Here is the call graph for this function:



Here is the caller graph for this function:

4.2.3.31 int CPU::stop_instr () [private]

STOP CPU Instruction. Acts as a breakpoint within the system. All execution is paused and user is ask to press return to continue.

Instruction length is 1 int [instr]

TODO: Document and check

Returns:

CPU return status such as interupts or OS calls

Here is the caller graph for this function:

4.2.3.32 int CPU::stor_instr () [private]

STOR CPU Instruction. Pops a valud from the stack and stores it in memory.

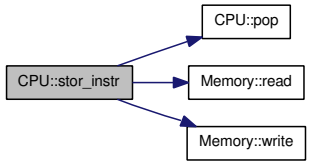
1. Pops value from stack
2. Reads immediate virtual address from instruction memory
3. Writes value to memory at address

Instruction length is 2 int [instr][address]

Returns:

CPU return status such as interupts or OS calls

Here is the call graph for this function:



Here is the caller graph for this function:

4.2.3.33 int CPU::sub_instr () [private]

SUB CPU Instruction. Subtracts 2nd to top value on stack form the top value

1. Pops 1st value from stack

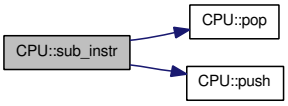
2. Pops 2nd value from stack
3. Subtracts 2nd from 1st value
4. Pushes answer back onto stack

Instruction length is 1 int [instr]

Returns:

CPU return status such as interupts or OS calls

Here is the call graph for this function:



Here is the caller graph for this function:

4.2.3.34 int CPU::tlb_have_entry (mem_addr_t *virtual_addr*) [private]

Does the TLB have an entry for the virtual address.

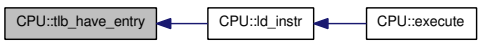
Parameters:

virtual_addr Virtual address being translated

Returns:

boolean

Here is the caller graph for this function:



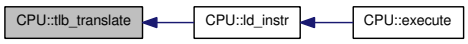
4.2.3.35 mem_addr_t CPU::tlb_translate (mem_addr_t *virtual_addr*) [private]

Translate a virtual address to a physical address.

Returns:

physical address or 0 if not available (tlb fault)

Here is the caller graph for this function:



4.2.3.36 int CPU::unlock_instr () [private]

UNLOCK CPU Instruction. Releases lock from LOCK instruction.

Instruction length is 1 int [instr]

Returns:

CPU return status such as interupts or OS calls

Here is the caller graph for this function:

4.2.4 Member Data Documentation

4.2.4.1 Process* CPU::cur_proc [private]

Current process.

4.2.4.2 mem_addr_t CPU::ip [private]

Instruction pointer.

4.2.4.3 Memory* CPU::mem [private]

Memory attached to this CPU.

4.2.4.4 int CPU::offset [private]

To support memory relocation in processes until VMM is enabled.

4.2.4.5 int CPU::reg[REGISTERSIZE] [private]

4.2.4.6 int CPU::sp [private]

Stack pointer: position within current process stack.

4.2.4.7 int CPU::stack[STACKSIZE] [private]

The documentation for this class was generated from the following files:

- CPU.h
- CPU.cpp