**Register Machine Model in Scala**

Contrast this to stack machine in [VM for Expressions](http://lara.epfl.ch/web2010/cc09:vm_for_expressions)

[sealed](http://scala-lang.org) [abstract](http://scala-lang.org) [class](http://scala-lang.org) Value

[case](http://scala-lang.org) [class](http://scala-lang.org) Immediate(const : Int) [extends](http://scala-lang.org) Value

[sealed](http://scala-lang.org) [abstract](http://scala-lang.org) [class](http://scala-lang.org) Address [extends](http://scala-lang.org) Value

[case](http://scala-lang.org) [class](http://scala-lang.org) Register(r : Int) [extends](http://scala-lang.org) Address

[case](http://scala-lang.org) [class](http://scala-lang.org) Memory(a : Int) [extends](http://scala-lang.org) Address

[case](http://scala-lang.org) [class](http://scala-lang.org) Indirect(r : Int) [extends](http://scala-lang.org) Address

[case](http://scala-lang.org) [class](http://scala-lang.org) IndirectOffset(r : Int, offset : Int) [extends](http://scala-lang.org) Address

[sealed](http://scala-lang.org) [abstract](http://scala-lang.org) [class](http://scala-lang.org) ArithOp

[case](http://scala-lang.org) [object](http://scala-lang.org) Mul [extends](http://scala-lang.org) ArithOp

[case](http://scala-lang.org) [object](http://scala-lang.org) Add [extends](http://scala-lang.org) ArithOp

[case](http://scala-lang.org) [object](http://scala-lang.org) Sub [extends](http://scala-lang.org) ArithOp

[sealed](http://scala-lang.org) [abstract](http://scala-lang.org) [class](http://scala-lang.org) Instruction

[case](http://scala-lang.org) [class](http://scala-lang.org) Quad(destination : Address,

op : ArithOp,

source1 : Value,

source2 : Value) [extends](http://scala-lang.org) Instruction

/\* Possible restrictions to above, on various architectures:

- source1,source2 must be registers (use Move if needed)

- destination = source1

- indirect addressing only for some registers

- register 0 always stores constant 0

\*/

[case](http://scala-lang.org) [class](http://scala-lang.org) Move(destination : Address,

source : Value) [extends](http://scala-lang.org) Instruction

[case](http://scala-lang.org) [class](http://scala-lang.org) BranchIfZero(r : Int, pcOffset : Int) [extends](http://scala-lang.org) Instruction

[case](http://scala-lang.org) [class](http://scala-lang.org) BranchIndirect(r : Int) [extends](http://scala-lang.org) Instruction

[case](http://scala-lang.org) [class](http://scala-lang.org) JumpSubroutine(pcOffset : Int) [extends](http://scala-lang.org) Instruction

[class](http://scala-lang.org) Computer([var](http://scala-lang.org) code : Array[Instruction],

[var](http://scala-lang.org) pc : Int, // program counter (current instruction)

[var](http://scala-lang.org) reg : Array[Int], // registers

[var](http://scala-lang.org) mem : Array[Int] // memory

)

{

[def](http://scala-lang.org) compute(op : ArithOp, x : Int, y : Int) : Int = op [match](http://scala-lang.org) {

[case](http://scala-lang.org) Mul => x \* y

[case](http://scala-lang.org) Add => x + y

[case](http://scala-lang.org) Sub => x - y

}

[def](http://scala-lang.org) load(v : Value) : Int = v [match](http://scala-lang.org) {

[case](http://scala-lang.org) Immediate(const) => const

[case](http://scala-lang.org) Register(r) => reg(r)

[case](http://scala-lang.org) Memory(a) => mem(a)

[case](http://scala-lang.org) Indirect(r) => mem(reg(r))

[case](http://scala-lang.org) IndirectOffset(r,offset) => mem(reg(r) + offset)

}

[def](http://scala-lang.org) store(a : Address, x : Int) = a [match](http://scala-lang.org) {

[case](http://scala-lang.org) Register(r) => { reg(r) = x }

[case](http://scala-lang.org) Memory(a) => { mem(a) = x }

[case](http://scala-lang.org) Indirect(r) => { mem(reg(r)) = x }

[case](http://scala-lang.org) IndirectOffset(r,offset) => { mem(reg(r) + offset) = x }

}

[def](http://scala-lang.org) step = code(pc) [match](http://scala-lang.org) {

[case](http://scala-lang.org) Quad(destination,op,source1,source2) => {

store(destination, compute(op, load(source1), load(source2)))

pc = pc + 1

}

[case](http://scala-lang.org) Move(destination, source) => {

store(destination, load(source))

pc = pc + 1

}

[case](http://scala-lang.org) BranchIfZero(r, pcOffset) => {

[if](http://scala-lang.org) (reg(r)==0)

pc = pc + pcOffset

[else](http://scala-lang.org)

pc = pc + 1

}

[case](http://scala-lang.org) BranchIndirect(r) => { pc = reg(r) }

[case](http://scala-lang.org) JumpSubroutine(pcOffset) => {

reg(15) = pc + 1

pc = pc + pcOffset

}

}

[def](http://scala-lang.org) run = {

[while](http://scala-lang.org) (pc < code.length)

step

}

}

[object](http://scala-lang.org) Test {

/\* Memory layout:

0 :

...

100 : x (initially 7)

104 : y (initially 2)

108 : z (initially 6)

Execute statement:

x = x\*y + y\*z + x\*z

Result should be 68.

\*/

[val](http://scala-lang.org) xAddr = 100

[val](http://scala-lang.org) yAddr = 104

[val](http://scala-lang.org) zAddr = 108

[def](http://scala-lang.org) initMem() : Array[Int] = {

[val](http://scala-lang.org) memory = [new](http://scala-lang.org) Array[Int](2048)

memory(xAddr) = 7

memory(yAddr) = 2

memory(zAddr) = 6

memory

}

[def](http://scala-lang.org) main(args : Array[String]) = {

[val](http://scala-lang.org) registers = [new](http://scala-lang.org) Array[Int](32)

// heavy use of memory addressing

[val](http://scala-lang.org) codeMemory : Array[Instruction] = List(

Quad(Register(1), Mul, Memory(xAddr), Memory(yAddr)),

Quad(Register(2), Mul, Memory(yAddr), Memory(zAddr)),

Quad(Register(1), Add, Register(1), Register(2)),

Quad(Register(2), Mul, Memory(xAddr), Memory(zAddr)),

Quad(Memory(xAddr), Add, Register(1), Register(2))

).toArray

[var](http://scala-lang.org) computer = [new](http://scala-lang.org) Computer(codeMemory,0,registers, initMem)

computer.run

println(computer.mem(xAddr))

// all arithmetic operations use registers,

// always fresh register unless value stored previously

[val](http://scala-lang.org) codeManyReg : Array[Instruction] = List(

Move(Register(1), Memory(xAddr)),

Move(Register(2), Memory(yAddr)),

Move(Register(3), Memory(zAddr)),

Quad(Register(4), Mul, Register(1), Register(2)),

Quad(Register(5), Mul, Register(2), Register(3)),

Quad(Register(6), Mul, Register(1), Register(3)),

Quad(Register(7), Add, Register(4), Register(5)),

Quad(Register(8), Add, Register(7), Register(6)),

Move(Memory(xAddr), Register(8))

).toArray

computer = [new](http://scala-lang.org) Computer(codeManyReg,0,registers, initMem)

computer.run

println(computer.mem(xAddr))

// all arithmetic operations use registers,

// minimizing memory traffic and registers

[val](http://scala-lang.org) codeSmart : Array[Instruction] = List(

Move(Register(1), Memory(xAddr)),

Move(Register(2), Memory(yAddr)),

Move(Register(3), Memory(zAddr)),

Quad(Register(4), Mul, Register(1), Register(2)),

Quad(Register(5), Mul, Register(2), Register(3)),

Quad(Register(4), Add, Register(4), Register(5)),

Quad(Register(5), Mul, Register(1), Register(3)),

Quad(Register(4), Add, Register(4), Register(5)),

Move(Memory(xAddr), Register(4))

).toArray

computer = [new](http://scala-lang.org) Computer(codeSmart,0,registers, initMem)

computer.run

println(computer.mem(xAddr))

// Question: can we do it with only one memory load for each x,y,z

// but with even fewer registers used?

[val](http://scala-lang.org) codeFourReg : Array[Instruction] = List(

Move(Register(1), Memory(xAddr)),

Move(Register(2), Memory(yAddr)),

Move(Register(3), Memory(zAddr)),

Quad(Register(4), Mul, Register(1), Register(2)),

Quad(Register(2), Mul, Register(2), Register(3)),

Quad(Register(4), Add, Register(4), Register(2)),

Quad(Register(1), Mul, Register(1), Register(3)),

Quad(Register(4), Add, Register(4), Register(1)),

Move(Memory(xAddr), Register(4))

).toArray

computer = [new](http://scala-lang.org) Computer(codeFourReg,0,registers, initMem)

computer.run

println(computer.mem(xAddr))

}

}