Contents

1 README 2

2 CPU.hdl 3

3 Computer.hdl 5

4 CpuMul.hdl 6

5 ExtendAlu.hdl 8

6 Memory.hdl 9

1 README

```
nivkeren,ransha
===============================================================================
Niv Keren, ID 201478351, niv.keren@mail.huji.ac.il
Ran Shaham, ID 203781000, ran.shaham1@mail.huji.ac.il
===============================================================================

                           Project 5 - Computer Architecture
                           ---------------------------------


Submitted Files
---------------
README        - This file.
CPU.hdl        - Our implementation of the CPU.
ExtendAlu.hdl    - ALU with multiplication and shifts.
CpuMul.hdl    - CPU that uses the ExtendAlu.
Memory.hdl    - Encapsulates 16K RAM, screen memory and keyboard register.
Computer.hdl    - Combines CPU, Memory and ROM to a working computer!

Remarks
-------
* Our implementation followed the instructions given in the lectures, which
  were simple and bold.
```

2 CPU.hdl

```
// This file is part of www.nand2tetris.org
// and the book "The Elements of Computing Systems"
// by Nisan and Schocken, MIT Press.
// File name: projects/05/CPU.hdl

/**
 * The Hack CPU (Central Processing unit), consisting of an ALU,
 * two registers named A and D, and a program counter named PC.
 * The CPU is designed to fetch and execute instructions written in
 * the Hack machine language. In particular, functions as follows:
 * Executes the inputted instruction according to the Hack machine
 * language specification. The D and A in the language specification
 * refer to CPU-resident registers, while M refers to the external
 * memory location addressed by A, i.e. to Memory[A]. The inM input
 * holds the value of this location. If the current instruction needs
 * to write a value to M, the value is placed in outM, the address
 * of the target location is placed in the addressM output, and the
 * writeM control bit is asserted. (When writeM==0, any value may
 * appear in outM). The outM and writeM outputs are combinational:
 * they are affected instantaneously by the execution of the current
 * instruction. The addressM and pc outputs are clocked: although they
 * are affected by the execution of the current instruction, they commit
 * to their new values only in the next time step. If reset==1 then the
 * CPU jumps to address 0 (i.e. pc is set to 0 in next time step) rather
 * than to the address resulting from executing the current instruction.
 */

CHIP CPU {

    IN  inM[16],         // M value input  (M = contents of RAM[A])
        instruction[16], // Instruction for execution
        reset;           // Signals whether to re-start the current
                         // program (reset==1) or continue executing
                         // the current program (reset==0).

    OUT outM[16],        // M value output
        writeM,          // Write to M?
        addressM[15],    // Address in data memory (of M)
        pc[15];          // address of next instruction

    PARTS:
        // C-Instruction decoding
        And16(a=instruction, b=true,  out[15]=opcode, out[12]=ca,
          out[11]=c1, out[10]=c2, out[9] =c3, out[8]=c4, out[7]=c5,
              out[6] =c6, out[5]=writeA, out[4]=writeD, out[3]=writeM1,
          out[2]=j1, out[1] =j2, out[0] =j3);

    // writeM is true iff it is a C-instruction and d3==1
        And(a=writeM1, b=opcode, out=writeM);

        // A Register logic
        Mux16(a=instruction, b=aluout, sel=opcode, out=aregin);
    Not(in=opcode, out=negopcode);
    // Load address to ARegister if it is an A-Instruction or d1==1
    Or(a=negopcode, b=writeA, out=aload);
    ARegister(in=aregin, load=aload, out=aregout, out[0..14]=addressM);

    // D Register logic - similar to A Register
    And(a=writeD, b=opcode, out=dload);
```

```
        DRegister(in=aluout, load=dload, out=dregout);

    // ALU logic
    // The a-bit in the C-Instruction determines whether to handle
    // A value or M value. This bit is extracted in the C-Instruction
    // decoding at the beginning of the chip
    Mux16(a=aregout, b=inM, sel=ca, out=aorm);
    ALU(x=dregout, y=aorm, zx=c1, nx=c2, zy=c3, ny=c4, f=c5, no=c6,
            zr=zr, ng=ng, out=aluout, out=outM);

    // PC logic
    Not(in=ng, out=nng);
    Not(in=zr, out=nzr);
    And(a=nng, b=nzr, out=pt);

    And(a=j3, b=pt, out=jgt);
    And(a=j1, b=ng, out=jlt);
    And(a=j2, b=zr, out=jeq);
    And(a=j1, b=j2, out=tmp);
    And(a=tmp, b=j3, out=jmp);
    // This covers every jump condition (e.g jge is satisfied by jgt or
    // jeq)
    Or8Way(in[0]=jmp, in[1]=jgt, in[2]=jlt, in[3]=jeq, in[4..7]=false,
           out=pcload1);
    And(a=pcload1, b=opcode, out=pcload);

    PC(in=aregout, load=pcload, inc=true,
              reset=reset, out[0..14]=pc);
}
```

3 Computer.hdl

```
// This file is part of www.nand2tetris.org
// and the book "The Elements of Computing Systems"
// by Nisan and Schocken, MIT Press.
// File name: projects/05/Computer.hdl

/**
 * The HACK computer, including CPU, ROM and RAM.
 * When reset is 0, the program stored in the computer's ROM executes.
 * When reset is 1, the execution of the program restarts.
 * Thus, to start a program's execution, reset must be pushed "up" (1)
 * and "down" (0). From this point onward the user is at the mercy of
 * the software. In particular, depending on the program's code, the
 * screen may show some output and the user may be able to interact
 * with the computer via the keyboard.
 */

CHIP Computer {

    IN reset;

    PARTS:
        // Exactly as shown in class: Unit 5.4, 25'40''
        CPU(inM=memout, instruction=romout, reset=reset, writeM=writeM,
        outM=outM, addressM=addressM, pc=pc);
    ROM32K(address=pc, out=romout);
    Memory(in=outM, address=addressM, load=writeM, out=memout);
}
```

4 CpuMul.hdl

```
/**
* This chip is an extension of the book CPU by using the extended ALU.
* More specificly if instruction[15]==0 or (instruction[14] and instruction[13] equals 1)
* the CpuMul behave exactly as the book CPU.
* While if it is C instruction and instruction[13] == 0 the output will be D*A/M
* (according to instruction[12]).
* Moreover, if it is c instruction and instruction[14] == 0 it will behave as follows:
*
* instruction:  | 12 | 11 | 10 |
* _____________________________
* shift left D  | 0  | 1  | 1  |
* shift left A  | 0  | 1  | 0  |
* shift left M  | 1  | 1  | 0  |
* shift right D | 0  | 0  | 1  |
* shift right A | 0  | 0  | 0  |
* shift right M | 1  | 0  | 0  |
**/

CHIP CpuMul{

    IN  inM[16],         // M value input  (M = contents of RAM[A])
        instruction[16], // Instruction for execution
        reset;           // Signals whether to re-start the current
                         // program (reset=1) or continue executing
                         // the current program (reset=0).

    OUT outM[16],        // M value output
        writeM,          // Write into M?
        addressM[15],    // Address in data memory (of M)
        pc[15];          // address of next instruction

     PARTS:
    // The same as CPU except for the ALU part, which is more intuitive.

        // C-Instruction decoding
        And16(a=instruction, b=true,  out[15]=opcode, out[12]=ca,
          out[5]=writeA, out[4]=writeD, out[3]=writeM1,
          out[2]=j1, out[1] =j2, out[0] =j3);

    // writeM is true iff it is a C-instruction and d3==1
        And(a=writeM1, b=opcode, out=writeM);

        // A Register logic
        Mux16(a=instruction, b=aluout, sel=opcode, out=aregin);
    Not(in=opcode, out=negopcode);
    // Load address to ARegister if it is an A-Instruction or d1==1
    Or(a=negopcode, b=writeA, out=aload);
    ARegister(in=aregin, load=aload, out=aregout, out[0..14]=addressM);

    // D Register logic - similar to A Register
    And(a=writeD, b=opcode, out=dload);
        DRegister(in=aluout, load=dload, out=dregout);

    // ALU logic
    // The a-bit in the C-Instruction determines whether to handle
    // A value or M value. This bit is extracted in the C-Instruction
    // decoding at the beginning of the chip
    Mux16(a=aregout, b=inM, sel=ca, out=aorm);
    ExtendAlu(x=dregout, y=aorm, instruction=instruction[6..14],
            zr=zr, ng=ng, out=aluout, out=outM);

    // PC logic
    Not(in=ng, out=nng);
    Not(in=zr, out=nzr);
    And(a=nng, b=nzr, out=pt);

    And(a=j3, b=pt, out=jgt);
    And(a=j1, b=ng, out=jlt);
    And(a=j2, b=zr, out=jeq);
    And(a=j1, b=j2, out=tmp);
    And(a=tmp, b=j3, out=jmp);
    // This covers every jump condition (e.g jge is satisfied by jgt or
    // jeq)
    Or8Way(in[0]=jmp, in[1]=jgt, in[2]=jlt, in[3]=jeq, in[4..7]=false,
           out=pcload1);
    And(a=pcload1, b=opcode, out=pcload);

    PC(in=aregout, load=pcload, inc=true,
              reset=reset, out[0..14]=pc);
}
```

5 ExtendAlu.hdl

```
/**
* The input of the extends ALU is instruction[9] and x[16],y[16].
* the output is define as follows:
* If instruction[7..8] equals 1 the the output is exactly as the ALU.
* Where instruction[5]=zx,instruction[4]=nx,...,instruction[0]=no.
* If instruction[7] equals 0 the output will be x*y and disregard the rest
* of the instruction.
*
* If instruction[8] equals 0 the output will be shift.
* Then, if instruction[4] equals 0 it will return shift of y otherwise shift
* of x, moreover if instruction[5] equals 0 it will return shift right
* otherwise shift left.
**/
CHIP ExtendAlu{
     IN x[16],y[16],instruction[9];
     OUT out[16],zr,ng;

     PARTS:
         ALU(x=x, y=y, zx=instruction[5], nx=instruction[4],
              zy=instruction[3], ny=instruction[2],
              f =instruction[1], no=instruction[0],
              out=aluout);

    // Compute all shifts of x,y
    ShiftLeft(in=x, out=shiftxl);
    ShiftLeft(in=y, out=shiftyl);
    ShiftRight(in=x, out=shiftxr);
    ShiftRight(in=y, out=shiftyr);
    // Determine which shift to choose according to instruction[4..5]
    Mux4Way16(a=shiftyr, b=shiftxr, c=shiftyl, d=shiftxl,
          sel=instruction[4..5], out=shiftout);

    // Compute x*y
    Mul(a=x, b=y, out=mulout);

    // Determine the output according to instruction[7..8]
    Mux4Way16(a=mulout, b=shiftout, c=mulout, d=aluout,
          sel=instruction[7..8], out[15]=ng, out[0..7]=out07,
          out[8..15]=out815, out=out);

    // Same as ng check in the ALU
    Or8Way(in=out07, out=zr1);
    Or8Way(in=out815, out=zr2);
    Or(a=zr1, b=zr2, out=nzr);
    Not(in=nzr, out=zr);
}
```

6 Memory.hdl

```
// This file is part of www.nand2tetris.org
// and the book "The Elements of Computing Systems"
// by Nisan and Schocken, MIT Press.
// File name: projects/05/Memory.hdl

/**
 * The complete address space of the Hack computer's memory,
 * including RAM and memory-mapped I/O.
 * The chip facilitates read and write operations, as follows:
 *     Read:  out(t) = Memory[address(t)](t)
 *     Write: if load(t-1) then Memory[address(t-1)](t) = in(t-1)
 * In words: the chip always outputs the value stored at the memory
 * location specified by address. If load==1, the in value is loaded
 * into the memory location specified by address. This value becomes
 * available through the out output from the next time step onward.
 * Address space rules:
 * Only the upper 16K+8K+1 words of the Memory chip are used.
 * Access to address>0x6000 is invalid. Access to any address in
 * the range 0x4000-0x5FFF results in accessing the screen memory
 * map. Access to address 0x6000 results in accessing the keyboard
 * memory map. The behavior in these addresses is described in the
 * Screen and Keyboard chip specifications given in the book.
 */

CHIP Memory {
    IN in[16], load, address[15];
    OUT out[16];

    PARTS:
        // The two MSBs determine which memory block to load to
        DMux4Way(in=true, sel=address[13..14], a=ramload1, b=ramload2
                         , c=screenload1, d=ignore);
    Or(a=ramload1, b=ramload2, out=ramload3);

    // We only want to load a memory block if it is the chosen one by
    // the MSBs and load input == 1
        And(a=load, b=ramload3, out=ramload);
        RAM16K(in=in, address=address[0..13], load=ramload, out=ramout);

    And(a=load, b=screenload1, out=screenload);
    Screen(in=in, address=address[0..12], load=screenload, out=screenout);

    Keyboard(out=kbdout);

    // The MSBs also determine from which memory block to output
    Mux4Way16(a=ramout, b=ramout, c=screenout, d=kbdout,
          sel=address[13..14], out=out);
}
```