Project 2

Computer Processors (COMP1212)

This project involves implementing components of a computer processor covered in the videos. You should provide an implementation of each of the following components in hdl. You are provided with a description of the behaviour of each of the components as well as test files to ensure it behaves according to specification.

Implement the following .hdl files

- HalfAdder.hdl
- FullAdder.hdl
- Add16.hdl
- ALU.hdl
- Bit.hdl
- Register.hdl
- Inc16.hdl
- PC.hdl
- RAM8.hdl
- RAM64.hdl
- RAM512.hdl
- RAM4K.hdl
- RAM16K.hdl
- CPU.hdl
- Computer.hdl

Chip descriptions

Chip name: `HalfAdder`

Inputs: x, y

Outputs: sum, carry

Function: Computes the sum of two bits.

Chip name: `FullAdder`

Inputs: x, y, z

Outputs: sum, carry

Function: Computes the sum of three bits.

Chip name: `Add16`

Inputs: $x[16], y[16]$

Outputs: result

Function: Adds two 16-bit values. The most significant carry bit is ignored.

Chip name: `ALU`

Inputs: $x[16], y[16], zx, nx, zy, ny, f, no$

Outputs: out[16],zr,ng

Function: Computes one of the following functions: $x + y$, $x - y$, $y - x$, 0, 1, -1, x, y, $-x$, $-y$, $\neg x$, $\neg y$, $x + 1$, $y + 1$, $x - 1$, $y - 1$, $x \wedge y$, $x \vee y$ on two 16-bit inputs, according to 6 input bits denoted zx, nx, zy,ny, f, no. The bit-combinations that yield each function are documented Figure 2.6 of The elements of Computing Systems. In addition, the ALU computes two 1-bit outputs: if the ALU output is 0, zr is set to 1; otherwise zr is set to 0; If out < 0, ng is set to 1; otherwise ng is set to 0.

Chip name: `Bit`

Inputs: x, load

Outputs: out

Function: If load$[t] = 1$ then out$[t + 1] = x[t]$ else out does not change (out$[t + 1]$ = out$[t]$)

Chip name: `Register`

Inputs: x[16], load

Outputs: out[16]

Function: If load$[t] = 1$ then out$[t + 1] = x[t]$ else out does not change

Chip name: `Inc16`

Inputs: x[16]

Outputs: out[16]

Function: 16-bit incrementer: out $= x + 1$ (arithmetic addition)

Chip name: `PC`

Inputs: x[16], load, inc, reset

Outputs: out[16]

Function: A 16-bit counter with load and reset control bits.
if (reset[t] == 1) out[$t + 1$] = 0
else if (load[t] == 1) out[$t + 1$] = in[t]
else if (inc[t] == 1) out[$t + 1$] = out[t] + 1 (integer addition)
else out[$t + 1$] = out[t]

Chip name: `RAM8`

Inputs: x[16], load, address[3]

Outputs: out[16]

Function: Memory of 8 registers, each 16 bit-wide. 'Out' holds the value stored at the memory location specified by address. If load= 1, then the 'x' value is loaded into the memory location specified by address (the loaded value will be emitted to out after the next time step.)

Chip name: `RAM64`

Inputs: x[16], load, address[6]

Outputs: out[16]

Function: Memory of 64 registers, each 16 bit-wide. 'Out' holds the value stored at the memory location specified by address. If load= 1, then the 'x' value is loaded into the memory location specified by address (the loaded value will be emitted to out after the next time step.)

Chip name: `RAM512`

Inputs: x[16], load, address[6]

Outputs: out[16]

Function: Memory of 512 registers, each 16 bit-wide. Out holds the value stored at the memory location specified by address. If load=1, then the in value is loaded into the memory location specified by address (the loaded value will be emitted to out after the next time step.)

Chip name: `RAM4K`

Inputs: x[16], load, address[12]

Outputs: out[16]

Function: Memory of 4K registers, each 16 bit-wide. Out holds the value stored at the memory location specified by address. If load=1, then the in value is loaded into the memory location specified by address (the loaded value will be emitted to out after the next time step.)

Chip name: `RAM16K`

Inputs: x[16], load, address[14]

Outputs: out[16]

Function: Memory of 16K registers, each 16 bit-wide. Out holds the value stored at the memory location specified by address. If load=1, then the in value is loaded into the memory location specified by address (the loaded value will be emitted to out after the next time step.)

Chip name: `CPU`

Inputs: inM[16], instruction[16], reset

Outputs: outM[16], writeM ,addressM[16], pc[15]

Function: Consists of an ALU and a set of registers, 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 unit. If 'reset'= 1 then the CPU jumps to address 0 (i.e. sets 'pc'= 0 in next time unit) rather than to the address resulting from executing the current instruction.

Chip name: `Computer`

Inputs: reset

Outputs:

Function: 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.