Assignment 04 – CPU

EDB HDL

ANM, v04

Introduction

Now that the main building blocks (ALU, registers, programm counter) are implemented, it is time to complete the ALU.

What is missing?

- Decoder → Control logic for the muxer
- CPU integration

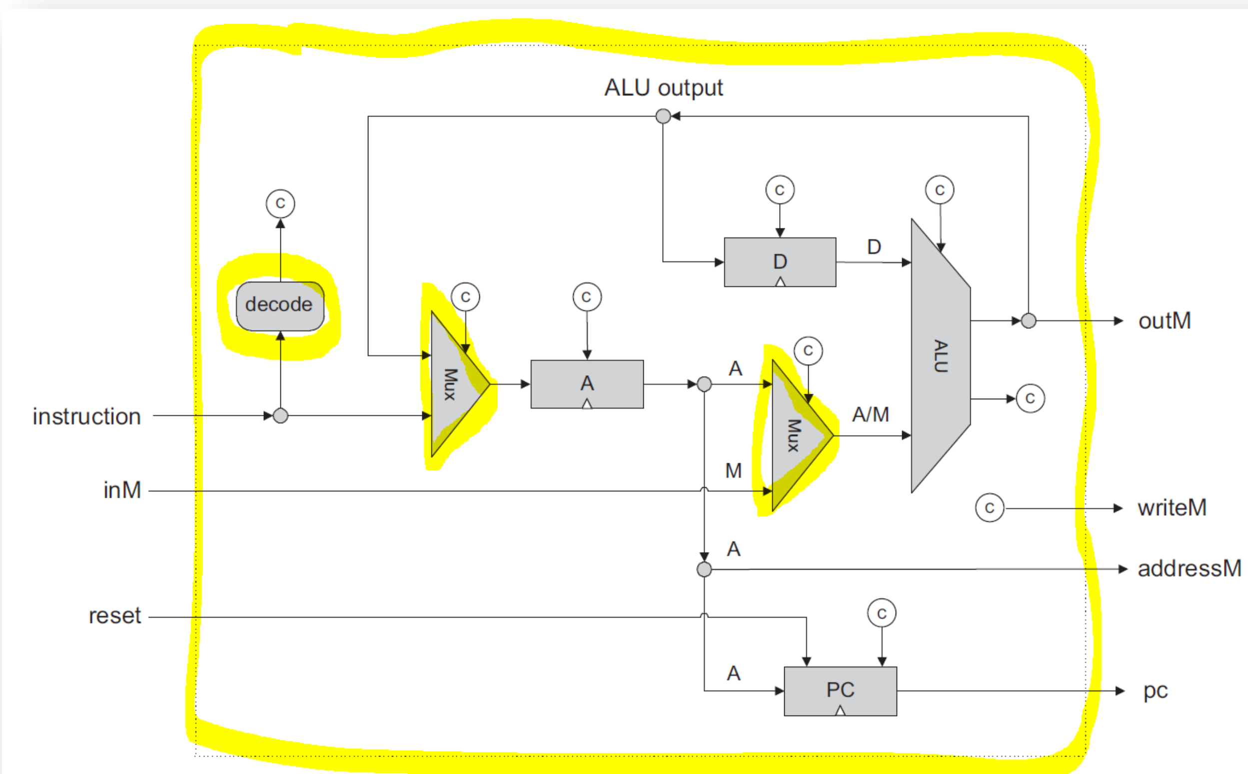

Figure 1: HACK CPU. The yellow blocks show the registers. Source: https://www.nand2tetris.org/

Instruction decoding

The program memory contains 16bit values, i.e. the commands. There are two types of commands supported: A-type and C-type.

The A-type commands are used to load the A register to a new value.

The C-type commands are used to control the operation of the ALU and the program counter.

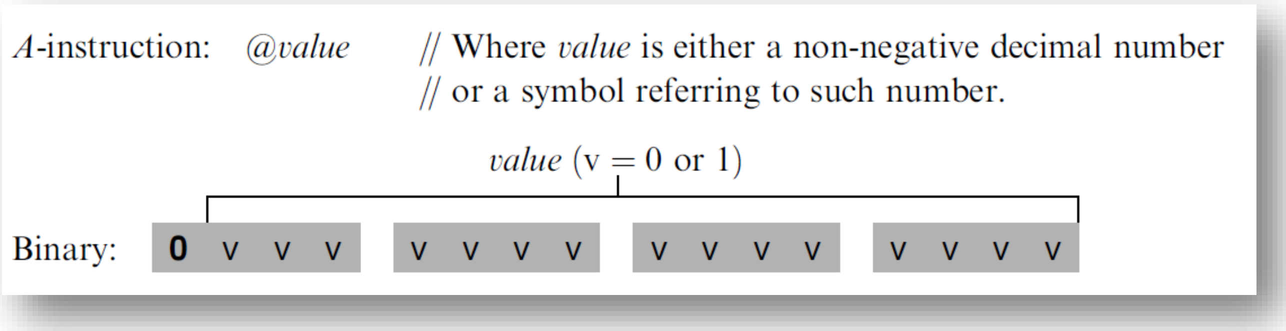

Figure 2: A-instruction. Source: https://www.nand2tetris.org/

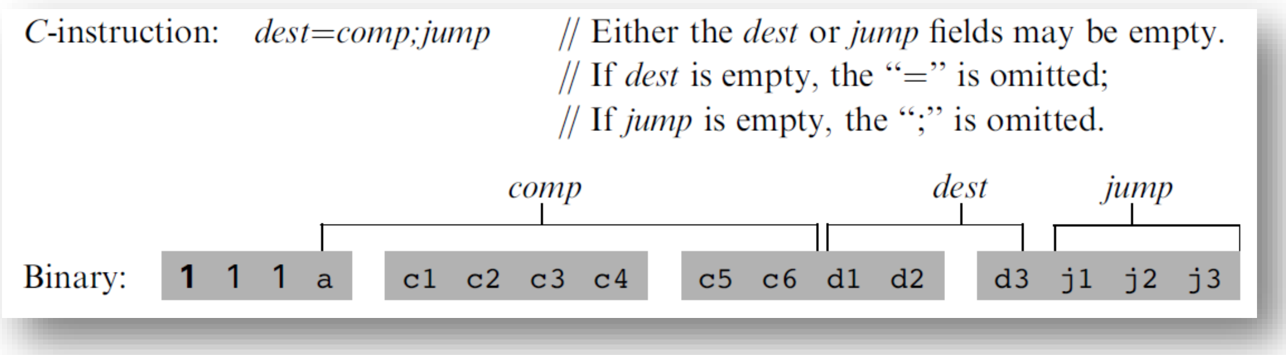

Figure 3: C-instruction. Source: https://www.nand2tetris.org/

Assembler language

The HACK assembler language uses commands that stand for different type-A or type-C commands as shown above.

- @constant → Load A to the value constant
- Computation specifier
- Destination specifier
- Jump specifier

comp (when a=0)	c1	c2	c3	c4	c5	c6	comp (when a=1)
0	1	0	1	0	1	0	
1	1	1	1	1	1	1	
-1	1	1	1	0	1	0	
D	0	0	1	1	0	0	
A	1	1	0	0	0	0	M
!D	0	0	1	1	0	1	
!A	1	1	0	0	0	1	!M
-D	0	0	1	1	1	1	
-A	1	1	0	0	1	1	-M
D+1	0	1	1	1	1	1	
A+1	1	1	0	1	1	1	M+1
D-1	0	0	1	1	1	0	
A-1	1	1	0	0	1	0	M-1
D+A	0	0	0	0	1	0	D+M
D-A	0	1	0	0	1	1	D-M
A-D	0	0	0	1	1	1	M-D
D&A	0	0	0	0	0	0	D&M
D|A	0	1	0	1	0	1	D|M

Figure 4: Computation specifier. Source: https://www.nand2tetris.org/

dest	d1	d2	d3
null	0	0	0
M	0	0	1
D	0	1	0
MD	0	1	1
A	1	0	0
AM	1	0	1
AD	1	1	0
AMD	1	1	1

jump	j1	j2	j3
null	0	0	0
JGT	0	0	1
JEQ	0	1	0
JGE	0	1	1
JLT	1	0	0
JNE	1	0	1
JLE	1	1	0
JMP	1	1	1

Figure 5: Destination and jump specifier. Source: https://www.nand2tetris.org/

Assembler command examples

In order to make the structure of the assembler language clearer a few examples are shown below:

Table 1: Examples of assembler commands.

Command	*Assembler*	*Instruction (refer to Figure 3 and Figure 2)*
Load A to the value 0x7FFF	@32767	**0**111_1111_1111_1111 (type A)
Load A to the value 0x0000	@0	**0**000_0000_0000_0000 (type A)
Set D to 1	D = 1	**1**11_0_111111_010_000 (type C) a = 0 c = 111111 (=1) d = 010 (destination is D) j = 000 (no jump)
Add D+A and store the result in D	D = D + A	**1**11_0_000010_010_000 a = 0 c = 000010 (addition) d = 010 (destination is D) j = 000 (no jump)
Increment A and store the result in the memory.	M=A+1	**1**11_0_110111_001_000 a = 0 c = 110111 (A+1) d = 001 (destination is M) j = 000 (no jump)

Tasks

- Create the folder structure
    - ./hack/sim
    - ./hack/src

Task 01 – Instruction decoder

The module instr_demux shall have the inputs and outputs as depicted in Figure 6.

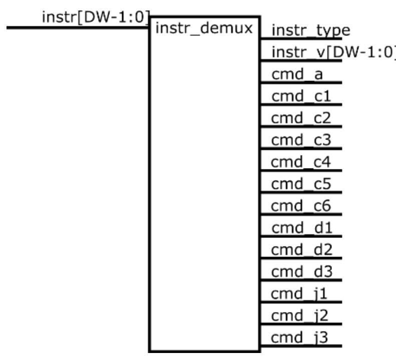

Figure 6: Block diagram of instr_demux.

- [1cp] Implement the **instr_demux** using combinatorial logic.
    - The command is specified by the input **instr**.
    - Use the MSB of instr to decode the instruction type (instr_type = instr[DW-1]).
    - instr_v is DW bits wide and contains the lower DW-1 bits of instr. Fill the leading bit with a 1'b0. You can output instr_v for type A and for type C instructions.
    - cmd_a … cmd_j3 are control bit outputs. Refer to Figure 3 for the exact location of each bit in the instr data. For type A instructions the commands need to be set to zero.
- [1cp] Create a testbench tb_instr_demux that checks the correctness of your implementation:
    - Use the commands in Table 1 in your testbench and check that the output is as expected.
    - Create a TCL script that controls the simulation and outputs a wave screen that allows to check the correct functionality of the CPU.

Task 02 – CPU

The CPU module is the heart of the HACK computer. It uses the previously implemented modules ALU, D-FF, PC and the instruction decoder from above.

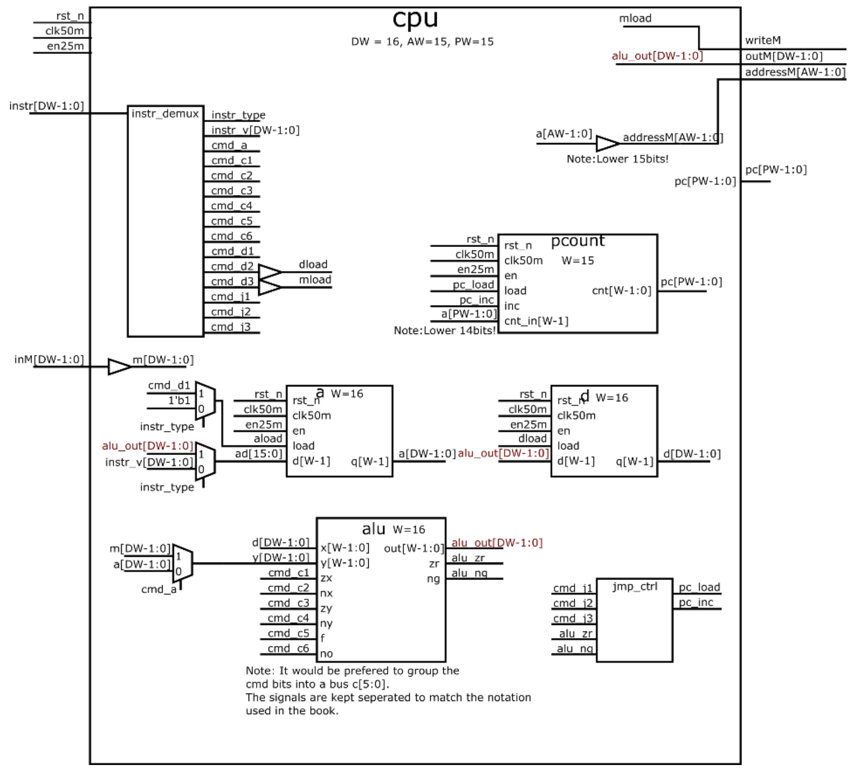

Figure 7: CPU block diagram.

- [1.5cp] Implement the module **cpu** as shown in Figure 7.
  The module *jmp_ctrl* controls the *load* and *inc* inputs of *pcount*. Refer to Figure 5 for a specification for the behavior.
- [1cp] Create a testbench tb_cpu that stimulates the input **instr** using the commands in Table 1 and check that the CPU outputs are as expected.

Task 03 – Documentation

- [0.5cp] Create a short summary report “doc_cpu.pdf” that shows the simulation result.
  - Show that your design fulfils the specification (verification).
  - Discuss why you chose your implementation method (advantages, disadvantages).