5. Computer Architecture

*Everything should be made simpler, but not too simple.*
*(Albert Einstein, 1879-1955)*

1. Overview

This chapter is work in progress. The purpose of the chapter is to give you enough information to build the Hack computer architecture. This architecture is based on the combinational and sequential chips that you built so far. We are now in a position to put all these pieces together, and create a working computer.

The chapter is based on a series of figures that describe various parts of the hardware. Since we want you to design and test the architecture on your own, we don't provide formal specifications. Instead, we give descriptive hardware sketches, without discussing the exact controls and data paths among the various chips. It is quite possible that in the course of building the computer you will feel that you need to build some chips that are not mentioned in our figures. That's OK.

FIGURE 5-1: The computer architecture. The data lines are 16-bit and the address lines are 15-bits. The small ovals represent control information that is emitted by some chips and used by other chips. This is two-cycle machine: in one clock cycle an instruction is fetched from memory, and in the next clock cycle the instruction is decoded and executed. The PC is incremented in one of these cycles, in preparation for the next fetch. The "fetch" and "execute" modes are generated by a device called 'F/E counter'.
1. The fetch / execute logic

The computer is a two-cycle machine which alternates continuously between two modes labeled "fetch" and "execute". The cycle is generated by a special chip called F/E counter, whose single-bit output (which is either f or e) is broadcast to many chips in the computer.

![Diagram of the fetch / execute instruction logic](image)

**FIGURE 5-2: The fetch / execute instruction logic.** The F/E counter, implemented as a single-bit counter, generates a continuous train of fetch / execute signals. Note that each instruction cycle requires two clock cycles.
2. Instruction decoding

When a 16-bit word emerges from the output lines of the memory, it can be either a data constant, or an address instruction, or a CSJ (Compute-Store-and-Jump) instruction. To be on the safe side, we assume that the 16-bits value can be all of the above. Thus, we send it to the ALU’s M input port, and, in the next clock cycle, we decode it twice: (a) as if it were an address command, and (b) as if it were a CSJ command. The decoded instruction fields are sent to other chips in the computer for processing (not shown in the figure). The chips either use or ignore this information, depending on the state in which they are, as dictated by their control bits.

**FIGURE 5-3: Instruction decoding logic.** Once the address of the current instruction is loaded into the address lines of the memory, the instruction bits appear at the output lines of the memory. In the next clock cycle, the instruction bits are stored in the instruction register. Next, the instruction is unpacked into its various fields, which are then sent to various parts of the computer. The instruction is also sent to the ALU’s M input port, but this operation is meaningless when the machine is in “fetch” mode.
3. The ALU

The ALU has two 16-bit inputs (the contents of the D register and the contents of the currently addressed memory location), and a single 16-bit output. In addition, the ALU has 6 input control bits (referred to collectively as \( \text{exp} \)) and two output control bits (referred to collectively as \( \text{zr/ng} \)). The ALU is a combinational chip which is aware of neither the clock nor the state in which the machine operates. At any given point of time, the ALU computes whatever function its control bits tell it to compute. The ALU output is then sent to various destinations in the machine. Depending on the state in which the machine operates (it has to be "execute") and other control information, the destinations either accept or ignore the ALU output.

**FIGURE 5-4: ALU execution and storage data paths.** The \( \text{exp} \) bits, which come from the decoding stage, tell the ALU which function to compute on M and D. The ALU output is then sent simultaneously to three destinations: the D register, the currently addressed memory location (M), and the A register. The decision of weather or not to accept the ALU output is done by every one of these destinations depending on the state of the machine and the \( \text{dest} \) enabling bits, which also come from the decoding stage. In the case of the A register, we also have to make sure that the instruction is of type CSJ, i.e. an instruction like \( A=... \), \( AD=... \), etc. This is done by the multiplexor that guards A's input port.
4. The Address register

The machine is equipped with three "containers": the A register, the D register, and the currently addressed memory location, labeled M. Unlike D and M, which may be effected only by CSJ commands, the A register can be effected either by a CSJ command or by an address command @addr, which is actually A=addr. Hence, the A register must be "open for business" in both command modes, and it must know from which source to select its new value.

FIGURE 5-5: A (address) register logic. Unlike D and M, which are effected only by CSJ commands, the A register can be loaded either by a CSJ command (inst=1) or by an address command (inst=0). The correct input channel is selected by a multiplexor that guards A's input port.
5. The Program Counter and the addressing logic

At any given clock cycle \( n \), the computer executes a certain instruction. At some point during the cycle, the program counter (PC) is set to contain the address of the instruction that must be fetched and executed at cycle \( n+1 \). This setting is done either implicitly, when the PC increments "by default", or explicitly, when the instruction contains a (materialized) jump specification.

With that in mind, the PC is a rather intricate chip. When we start the computer, it must be reset to 0. When the current instruction contains no jump specification, it must increment by 1. When the instruction contains a jump specification, and the ALU control bits indicate that the jump condition is satisfied, the PC register must be set to the value of the address register (A). This logic explains the left side of Figure 5-6.

**FIGURE 5-6:** PC updating logic (left) and addressing logic (right). Once the PC figures out the address of the next instruction, the right hand side of the figure is straightforward. In “fetch” mode, the multiplexor selects the address of the next instruction to be fetched and executed. In "execute" mode, the multiplexor selects the address of the memory location on which the ALU will operate. The contents of this location is called "M" in our terminology.
6. Cycle / state analysis

<table>
<thead>
<tr>
<th>addr</th>
<th>instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>@200</td>
</tr>
<tr>
<td>101</td>
<td>goto</td>
</tr>
<tr>
<td></td>
<td>...</td>
</tr>
<tr>
<td>200</td>
<td>D=1</td>
</tr>
</tbody>
</table>

![Diagram of cycle state analysis](image)

**FIGURE 5-7: Cycle / state analysis.** The table describes the sequence of state changes that occur when the computer runs the code given at the top left corner of the figure. The ??? question marks indicate meaningless, i.e. out-of-context information. The basic computer architecture is depicted for ease of reference.