Nirosh Ravindran

BSCP|CS|63|134

Computer systems 10.1P

Before explaining the **Fetch-Execute** let's first take a look at the program memory and the symbol table relevant to the Hack assembly program shown in Figure 1.

Program memory

Address | Instruction

----

0 | @i
1 | M=0
2 | @sum
3 | M=0
4 | @100
5 | D=A
6 | @n
7 | M=D
8 | (LOOP)
9 | @i
10 | D=M
11 | @n
12 | D=D-M
13 | @STOP
14 | D; JEQ
15 | @sum
16 | D=M
17 | @i
18 | M=M+1
19 | D=D+M
20 | @sum
21 | M=D
22 | @LOOP
23 | 0; JMP
24 | (STOP)
25 | @sum
26 | D=M
27 | @R1
28 | M=D

Symbol Table

R0	0
R1	1
R2	2
R3	3
…	…
R15	15
SCREEN	16384
KBD	24576
SP	0
LCL	1
ARG	2
THIS	3
THAT	4
LOOP	9
STOP	25
i	16
sum	17
n	18

The operations in the provided assembly code are mostly related to data manipulation between registers and memory. For these data transfers to occur within a computer architecture, buses are essential. Let's list the busses that are a part of this program:

Address Bus:

- The address bus is used during the fetch phase to specify the memory location from which an instruction is to be fetched.
- Examples in the code: **@i, @sum, @100, @n, @STOP, @LOOP,** etc.
- When these instructions are encountered, the corresponding memory locations are accessed using the address bus.

Data Bus:

- The data bus is utilized to transfer data between the CPU registers (A, D, and M registers) and memory.
- Examples in the code: **M=0, D=A, M=D, D=M, D=D-M, M=D,** etc.
- These instructions involve the transfer of data between registers and memory, facilitated by the data bus.

Control Bus:

- While not explicitly mentioned in the code, a control bus could be involved in signaling various control signals such as read, write, and jump conditions.
- Control signals are crucial for coordinating actions within the CPU and memory subsystem.

Now let's look at the main **Fetch-Execute** methods that takes place in the program

Fetch-Execute Cycle for @i (Address 0):

Fetching:

- The Program Counter (PC) is at address 0.
- It reads the instruction @i from the program memory.
- The instruction is loaded into the instruction register.

Execution:

- The instruction @i is an A-instruction, which means it sets the A register to a specific value "i".
- The value in the A register is set to the constant value associated with the symbol i, which is 0.
- After this cycle, the A register contains the value 0, which represents the memory location of variable i.

Fetch-Execute Cycle for D=M (Address 10):

Fetching:

- The Program Counter (PC) is at address 10.
- It reads the instruction D=M from the program memory.
- The instruction is loaded into the instruction register.

Execution:

- The instruction D=M is a computation instruction that retrieves the value stored in the memory location addressed by the value in the A register and stores it in the D register.
- The A register currently holds the value of i (0) because of the previous instruction @i.
- The value at memory location i (which is 0) is loaded into the D register.
- After this cycle, the D register now holds the initial value of i, which is 0.

Fetch-Execute Cycle for @n (Address 11):

Fetching:

- The Program Counter (PC) is at address 11.
- It reads the instruction @n from the program memory.
- The instruction is loaded into the instruction register.

Execution:

- The instruction @n is an A-instruction that loads the constant value 100 into the A register.
- The value 100 is now stored in the A register.
- The next instruction will use this value as an address to access memory.

Fetch-Execute Cycle for D=D-M (Address 12):

Fetching:

- The Program Counter (PC) is at address 12.
- It reads the instruction D=D-M from the program memory.
- The instruction is loaded into the instruction register.

Execution:

- The instruction D=D-M is a computation instruction that subtracts the value in the A register (which holds 100) from the value in the D register (which holds 0).
- The result of the subtraction (-100) is stored in the D register.
- After this cycle, the D register holds the result of the subtraction, which is -100.

Fetch-Execute Cycle for @STOP (Address 13):

Fetching:

- The Program Counter (PC) is at address 13.
- It reads the instruction @STOP from the program memory.
- The instruction is loaded into the instruction register.

Execution:

- The instruction @STOP is an A-instruction, meaning it sets the A register to a specific value (STOP in this case).
- The next instruction is not a computation or jump instruction; it's just a label (STOP). This instruction doesn't alter the control flow; it's a marker indicating the end of the loop.
- The program proceeds to the next instruction without making any changes to registers or memory because @STOP is not a computation or jump instruction.

Fetch-Execute Cycle for D; JEQ (Address 14):

Fetching:

- The Program Counter (PC) is at address 14.
- It reads the instruction D; JEQ from the program memory.
- The instruction is loaded into the instruction register.

Execution:

- The instruction D; JEQ is a conditional jump instruction. It checks the value in the D register and jumps to the specified label (STOP) if the condition is met.
- In this case, the condition is equality (JEQ), and the comparison is between the value in the D register and zero.
- If the value in the D register is zero (meaning D is equal to M in the previous D=D-M instruction), then the jump condition is true, and the Program Counter (PC) is updated to the address of the STOP label (24).
- However, in our scenario where D is -100 (as calculated in the previous instructions), the jump condition is false, and the program continues to the next instruction without jumping.

Now let's take a look at the function of the program after the jump condition is false:

- **@sum:** The A-instruction loads the value stored in the memory location labeled sum into the A register.
- **D=M:** The value in the A register (which is the content of sum) is loaded into the D register.
- **@i:** The A-instruction loads the value stored in the memory location labeled i into the A register.
- **M=M+1:** The value in the A register (which is the content of i) is incremented by 1. The result is stored back in the memory location labeled i.
- **D=D+M:** The value in the D register (which is the initial value of sum) is added to the updated value of i (incremented value). The result is stored in the D register.
- **@sum:** The A-instruction loads the memory location labeled sum into the A register.
- **M=D:** The value in the D register (which is the sum of the original sum and the incremented i) is stored back in the memory location labeled sum.
- **@LOOP:** The A-instruction loads the value of the label LOOP into the A register.

- **0; JMP:** This is an unconditional jump instruction (JMP) with the destination address set to 0. Essentially, it creates an infinite loop by jumping back to the address labeled LOOP.

- **(STOP):** This is a label indicating the end of the loop. Execution will continue sequentially until it reaches this point.

- **@sum:** The A-instruction loads the value stored in the memory location labeled sum into the A register.

- **D=M:** The value in the A register (which is the content of sum) is loaded into the D register.

- **@R1:** The A-instruction loads the value stored in the memory location labeled R1 into the A register.

- **M=D:** The value in the D register (which is the final sum of numbers 1 to 100) is stored in the memory location labeled R1. This effectively stores the result in register R1.

Let's consider a scenario where i starts at 99 and n is 100 at the beginning of the loop:

- **@i:** A-instruction loads the value in memory location i (which is 99) into the A register.
- **D=M:** D-register now holds the value 99.
- **@n:** A-instruction loads the value in memory location n (which is 100) into the A register.
- **D=D-M:** Subtracting 100 from 99 results in -1, and this value is stored in the D register.
- **@STOP:** A-instruction loads the value STOP into the A register. The jump condition (D; JEQ) checks if the result in the D register is zero.

In this scenario, the condition D; JEQ is not true because the result in the D register is -1, not zero. Therefore, the program will not jump to the STOP label, and it will continue with the next instruction in the loop.

The condition will become true when D becomes zero, meaning i is equal to n. In the case of the loop that sums numbers from 1 to 100, this will happen when i becomes 100. At that point, D will be zero after the subtraction, and the jump to STOP will occur, terminating the loop.

So, this is how the overall Hack CPU functions.