MEMORY & REGISTERS

As you go through the program, refer to the reference sheet at the end of this document to see where to go next when completing each step.

MEMORY

Below is a program stored in memory, beginning at memory address 0. **At each memory address, one byte (2 digits) of data is stored.** Memory addresses are the numbers on the top row.

0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	1E	1F	20	21	22
30	F2	00	00	00	01	30	F1	00	00	00	05	30	F0	00	00	00	00	62	11	71	00	00	00	22	60	10	61	21	70	00	00	00	12	00

REGISTERS

Our computer has 8 registers – 8 memory slots in the CPU – to hold onto the numbers we are currently working with. Like main memory, each register has an address, shown in the left row. The columns to the right of these addresses represent clock cycles that the computer goes through.

Address	Unless a new value is written to a register, the register keeps its previous value: copy these values from the previous clock cycle.													
0	0	0												
1	0	**5**												
2	**1**	1												
3	0	0												
4	0	0												
5	0	0												
6	0	0												
7	0	0												

Note: We have no register F. If you try to access register F, return a 0; if you're trying to write to register F, just move on. F is used as a flag to indicate we don't need to read or write to a register in a given instruction.

FETCH AND DECODE

Fetch and Decode will **ask for the instruction stored in memory**, starting at the address PC.
The value of PC will come from Execute. Parse the instruction as follows:

- the 1st hex digit is the **instruction code** (**iCd**)
- the 2nd hex digit is the **instruction function** (**iFn**)

If iCd = 3 or iCd = 6:

- the 3rd hex digit is **register A** (**rA**) and the 4th hex digit is **register B** (**rB**)

If iCd = 3 or iCd = 7:

- the next 8 hex digits (or 4 bytes) are called **valC**

And then calculate the following four values: (*NOTE: ALL VALUES ARE IN HEXADECIMAL*.)

If iCd is ___, then:	valP	srcA	srcB	dstE
0	PC	F (the hex value)	F (the hex value)	F (the hex value)
3	PC + 6	F (the hex value)	F (the hex value)	rB
6	PC + 2	rA	rB	rB
7	PC + 5	F (the hex value)	F (the hex value)	F (the hex value)

PC	0	6	C											
Instruction	30F2000 00001	30F10000 0005												
iCd	3	3												
iFn	0	0												
rA	F	F												
rB	2	1												
valC	1	5												
valP	6	C												
srcA	F	F												
srcB	F	F												
dstE	2	1												

EXECUTE

Execute works with a lot of the rest of the computer to make sure every instruction runs properly.

1. Start off by getting the following things: **iCd**, **iFn**, **valC, valP, srcA, srcB, and dstE** from **Fetch/Decode.**
2. **scrA** is the **register address** where the value you will use for **valA** is stored. Get this value from the register and do the same with **srcB** to get **valB**.
3. Send **iCd**, **iFn, valA, valB and valC** over to **ALU/DECIDE** and wait for two values back: **valE** and **bch**.
4. The value of **nextPC** will be **valP**, **with one exception**: **when iCd = 7 AND bch = 1**. In this case, we will move to somewhere else in our program, and nextPC will be valC. **Pass nextPC** to the PC cell in the next empty column of **Fetch/Decode.**
5. Lastly, tell Memory to **write the value valE to the register** with address **dstE**.

Get these values from Fetch/Decode (Step #1)														
iCd	3	3												
iFn	0	0												
valC	1	5												
valP	6	C												
srcA	F	F												
srcB	F	F												
dstE	2	1												
Get these values from registers (Step #2)														
valA	0	0												
valB	0	0												
Get these values from the ALU (Step #3)														
valE	1	5												
bch	0	0												
Calculate this value (Step #4)														
nextPC	6	C												
Finally, save valE back to the register at dtsE (Step #5)														

ALU AND DECIDE BRANCH

The job of the Arithmetic Logic Unit is to do the arithmetic and logic operations in the computer.

STEP 1 **Start by getting aluA and aluB**:

If iCd is:	aluA	aluB
0	0	0
3	valC	0
6	valA	valB
7	valC	0

STEP 2 **Calculate a new hexadecimal value, valE**:

If iCD is:	and iFn is __, then:	valE
~ 6	0, 1, 2	aluB + aluA
6	0	aluB + aluA
6	1	aluB - aluA
6	2	aluB ∧ aluA

STEP 3 Your job is also to figure out if the conditions are right for branching–Execute will then decide if we are actually going to branch.

If iCD is:	and iFn is __, then:	bch
~7	0, 1, 2	0
7	0	jump unconditionally: bch = 1
7	1	**If valE ≤ 0 in the previous clock cycle**, then bch = 1. **Otherwise**, bch = 0.
7	2	0

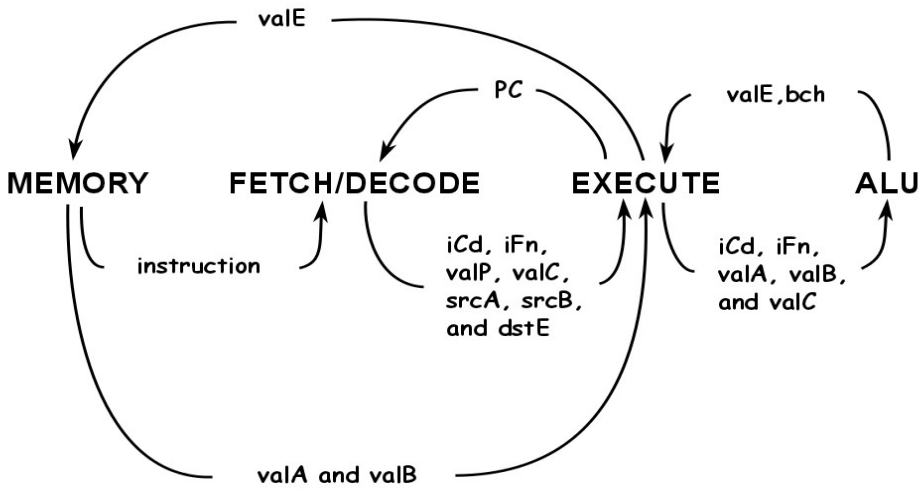
Get these values from Execute:														
iCd	3	3												
iFn	0	0												
valA	0	0												
valB	0	0												
valC	1	5												
Then calculate these:														
aluA	1	5												
aluB	0	0												
valE	1	5												
bch	0	0												

REFERENCE SHEET

Sequence of steps to be completed, including values to pass during a clock cycle:

1. An instruction is taken from memory at the address stored in PC.
2. Fetch/Decode analyzes the instruction and forwards information about the function of the instruction (iCd, iFn) and data (srcA, srcB, etc) to Execute.
3. Execute uses information from Fetch/Decode to get values from memory. Then, it forwards relevant data to the ALU to perform the correct computations.
4. ALU performs arithmetic operations and calculates a new value (valE). It also determines how the program will advance (bch)

5-7 Execute updates key registers (PC) and other relevant information as well as writes values into memory. Begin again at step 1.



Different instructions:

iCd	iFn	Does this:
0	0	Halts the computer
3	0	Moves a value into a register
6	0	Add
6	1	Subtract
6	2	Logical bitwise AND
7	0	Unconditional jump
7	1	Jump if less than or equal

Note: register A (rA) and register B (rB) are not used for the halt (00) instruction or the jump (7x) instructions