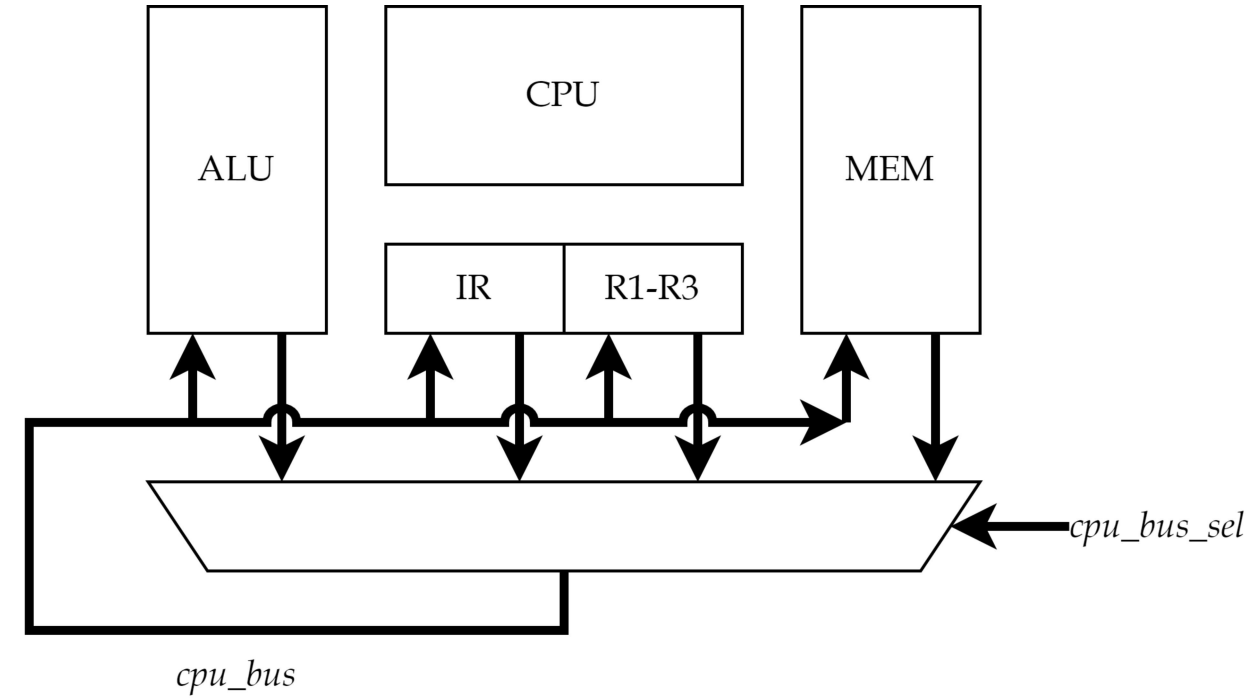
Introduction

In this problem, we are going to design a simple 8 bits CPU that can perform simple commands like ADD, COMPARE, SHIFT, JUMP, etc. The purpose of this problem is introducing you to designing process and implement the design using SystemVerilog.
Some main blocks of a CPU:

- CPU: The unit that controls the hole design, depends on command and state that are running.
- ALU: The calculating unit that performs the calculation like ADD, SHIFT, COMPARE, etc.
- IR: Instruction register that temporary store the ongoing instruction. See section Instruction register for more information about it's operation
- User register set: Set of registers that temporary hold data from the command that users can interact with. In this design, we have 4 registers that are marked as R0, R1, R2, R3.
  (The reason we have only 4 registers is because of the limitation of the bus size, see example in section ALU)
- Memory: Store instruction and data for the whole program.

The simplified blocks are in the figure below. Note that this design will be changed when we progress through the designing process.



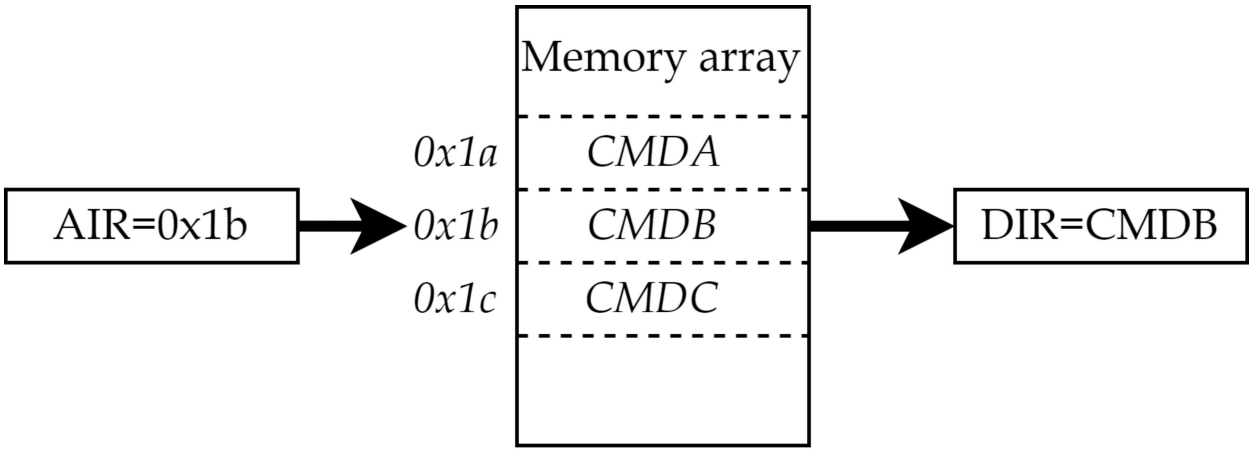
The signals *cpu_bus* and *cpu_bus_sel* are the data bus for the whole design and the selection of it's source. At one time, data from 1 block is selected and becomes the input for other blocks. This data can be captured to one ore more blocks depends on the CPU's decision.
The data size for this signal is 8 bits, and it is the size for all data and instructions in this design.

Instruction register

CPU use the instruction register (IR) to point to a Memory's address, extract data from it to process, and move to the new address.

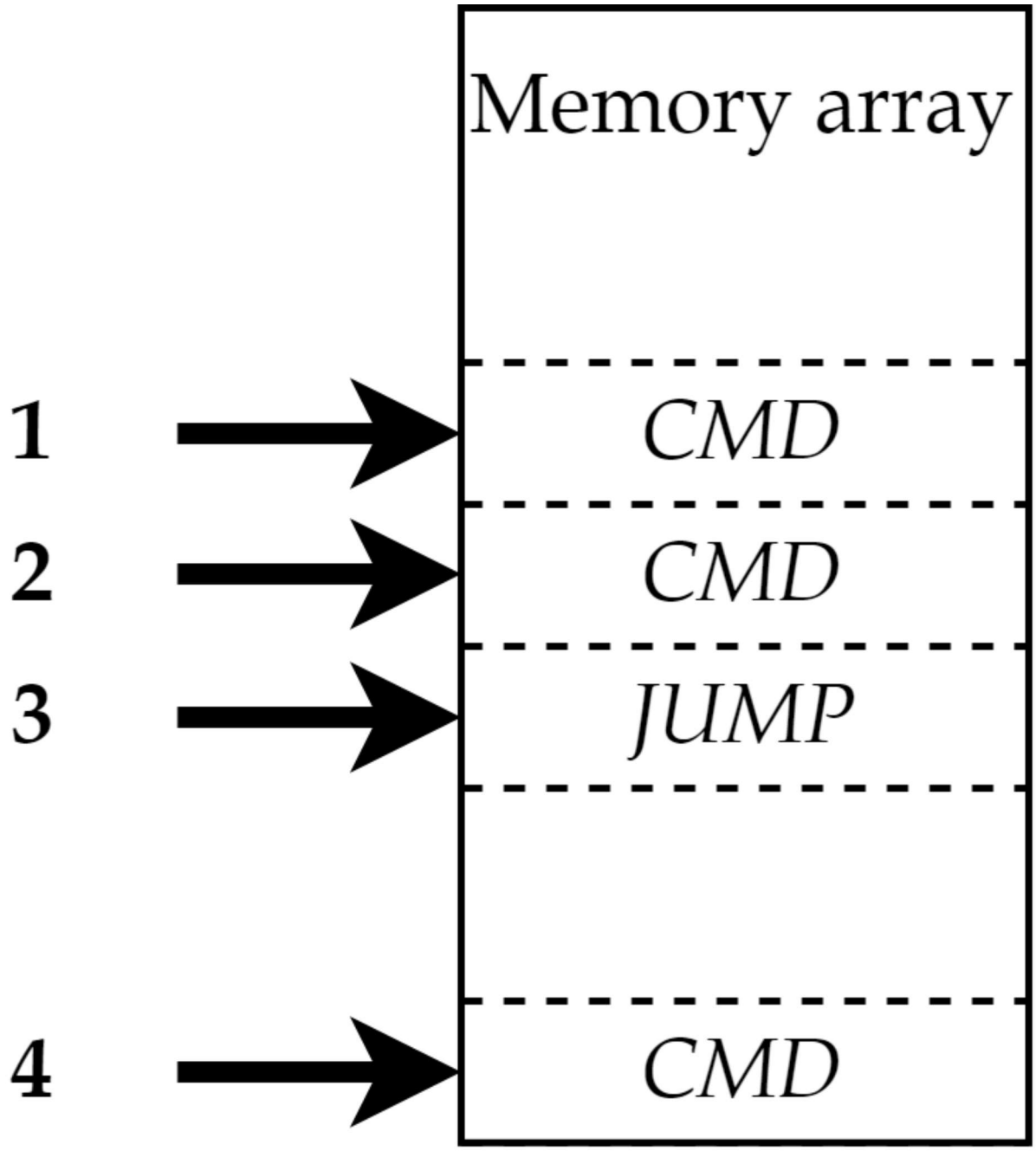
IR in this design is divided into 2 registers:

- Adress IR (AIR): Contain adress that the CPU wants to point to.
- Data IR (DIR): Contain data of the Memory that AIR point to. Image blelow shows an example of how AIR and DIR work.



In this part of document, both AIR and DIR are called IR for short description. They will be distinguised as we start to design the CPU.

During operation, IR normally increases the AIR pointer 1 at a time to sweep throughout the Memory's address. Sometimes the IR jumps to different addresses based on JUMP command's condition, see Command Jump section. Image below illustrates how the AIR points to Memory array in normal command and JUMP command.



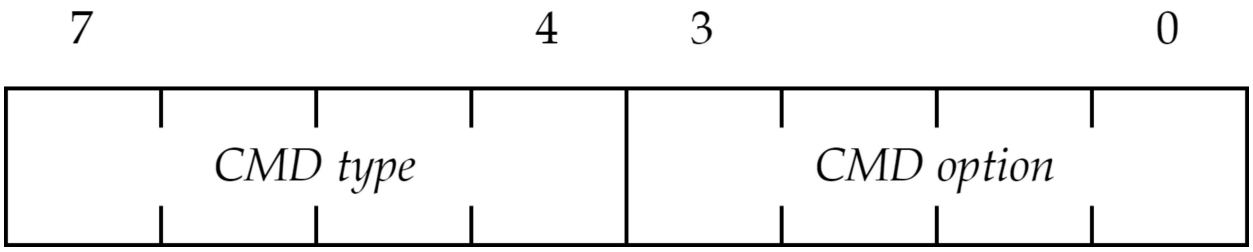
Note: Even though we see command as ADD, SHR, CMP, etc. as shown in next sections, the computer only looks at binary number in the DIR register to decide what is the command and what should be done. Thus, in the next sections, we will see how a command is stored as binary

Instruction Command set

There are 4 command types in this problem:

- ALU: Operators like ADD, SHIFT
- Load & Store: Functions to load and store data for registers and memory
- Jump: Jump the address of instruction pointer in the memory array
- Miscellaneous: Other commands like clear or end of program

As discussed before, instruction is 8 bits so command is separated to 2 regions: Command type in upper 4 bits and command option in lower 4 bits (see figure below). At one time, CPU takes 1 command from Memory and put to the IR. Then, the CPU uses the information from the IR to decide what operation it needs to do next.



The designing process for command set is not introduced in this document, it is infact another art in engineering world.

ALU

All ALU instructions use the command option (lower 4 bits) to select register. A command is divided as following description:

- Command type (ADD, SHIFT, etc.) takes 4 bits
- Register A takes 2 bits
- Register B takes 2 bits

All of the componnents use up 8 bits of the data size.

Register A (RA) & Register B (RB) are pointers that point to the user register set. They point to the user register set R0-R3 as described in examples below. And since we only have 2 bits, we can point only to 4 registers. That is the reason why we only have 4 registers in the user register set.
This also tell us that if we have bigger bus size, or we change the command structure, we may have more registers. However, choosing the right sizes for command and option is not a random work, and the process is not described in this document.

The command set for ALU type is listed in below table.

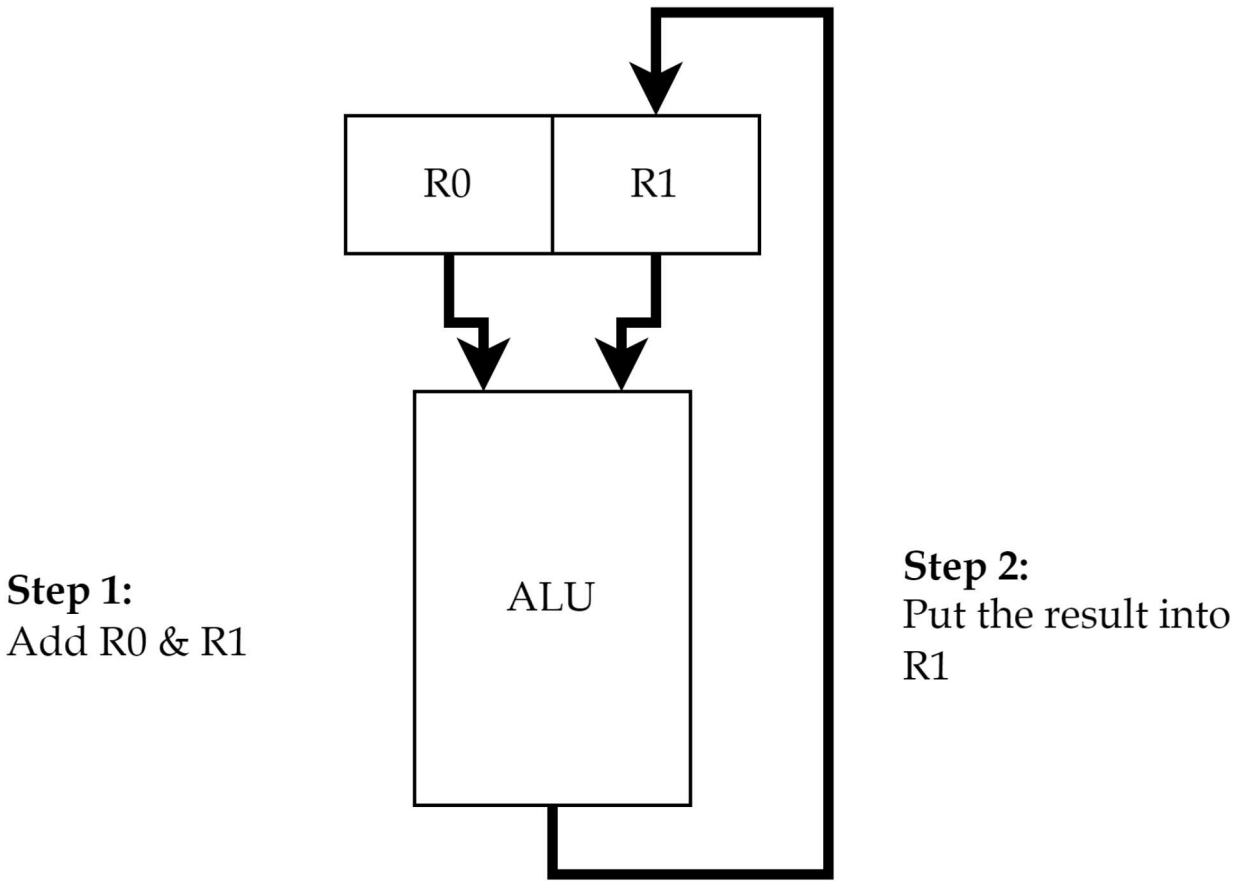
Command	Length	7	6	5	4	3:2	1:0	Description
ADD	1	1	0	0	0	RA	RB	ADD RA and RB then put result into RB

Command	Length	7	6	5	4	3:2	1:0	Description
SHR	1	1	0	0	1	RA	RB	SHIFT RA LEFT them put result into RB
SHL	1	1	0	1	0	RA	RB	SHIFT RA RIGHT them put result into RB
NOT	1	1	0	1	1	RA	RB	NOT RA then put result into RB
AND	1	1	1	0	0	RA	RB	AND RA with RB then put result into RB
OR	1	1	1	0	1	RA	RB	OR RA with RB then put result into RB
XOR	1	1	1	1	0	RA	RB	XOR RA with RB then put result into RB
CMP	1	1	1	1	1	RA	RB	COMPARE RA with RB

Example 1: Instruction Register (IR) has the value 0b10000001:

- This is ADD command
- RA is 0
- RB is 1

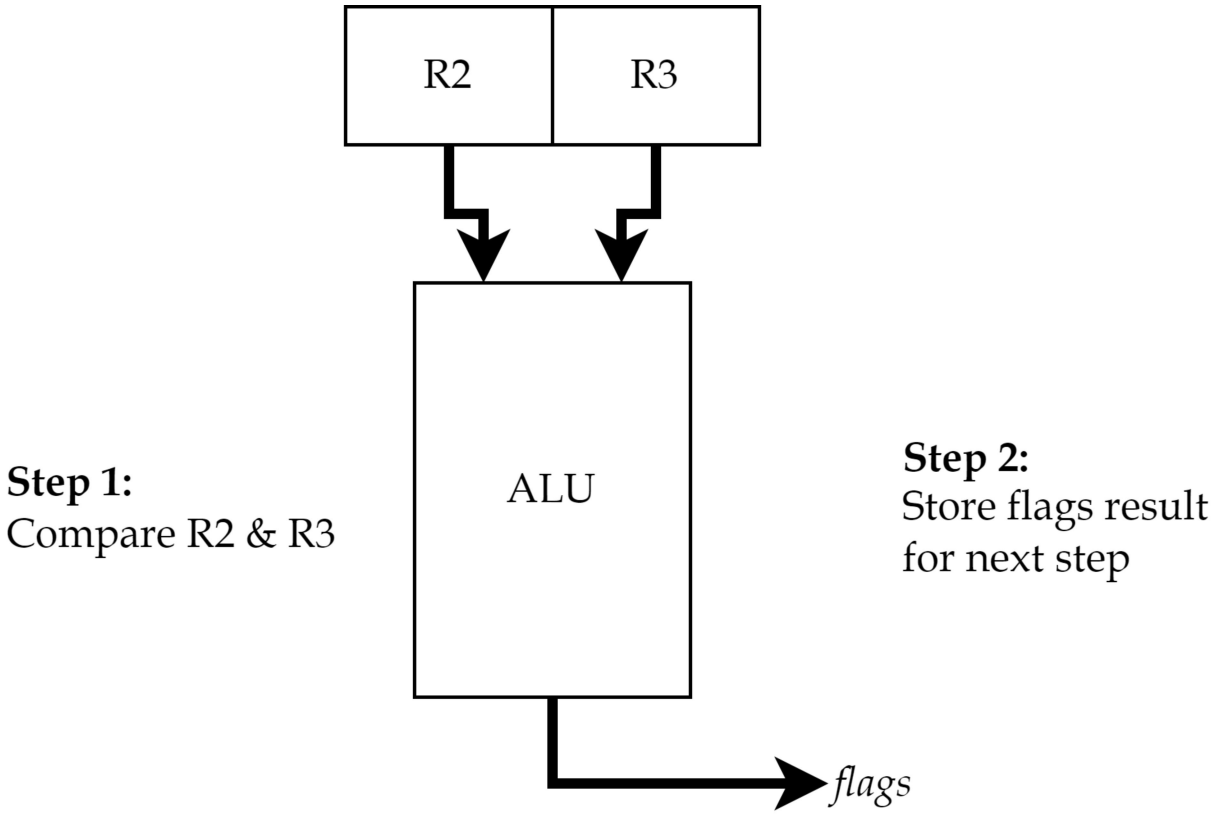
CPU will control the ALU to sum data in R0 and R1, then put the result to register R1. See image below.



Example 2: IR has the value 0b11111011

- This is COMPARE command
- RA is 2
- RB is 3

CPU will control the ALU to compare the data in register R2 and R3. See image below.



In this command, the result isn't stored in any registers (different from other command in ALU command set), but it is hold in ALU *flags*. These flags will be used for the JUMP command set. The flags can be *eq* if RA is equal to RB, or *al* if RA is larger than RB, etc. Please see section TBD.

Load & Store

This command set is used to load data into registers, or store their values to the memory.

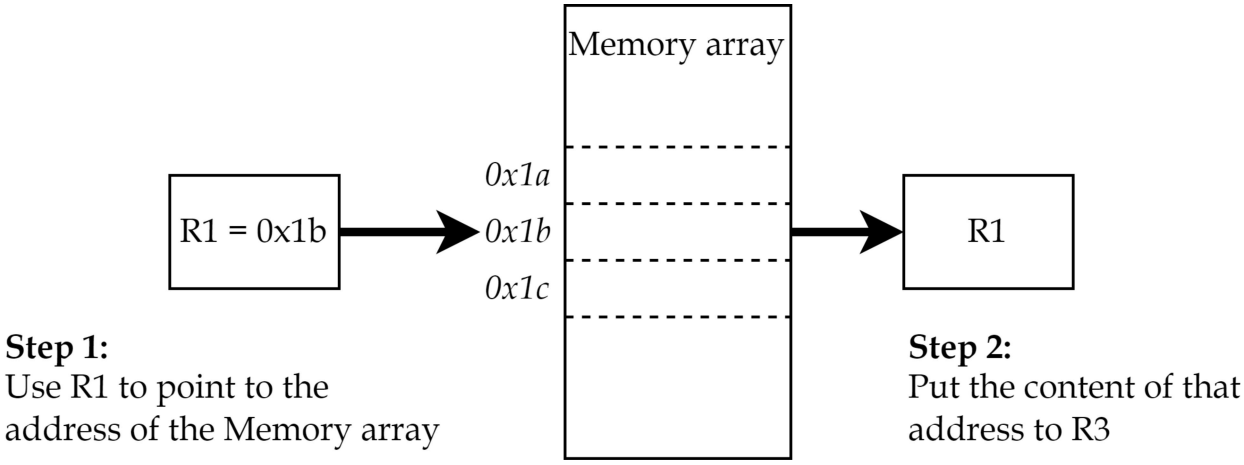
Command	Length	7	6	5	4	3:2	1:0	Description
LD	1	0	0	0	0	RA	RB	LOAD from RAM address in RA to register RB
ST	1	0	0	0	1	RA	RB	STORE RB to RAM address in RA
DATA	2	0	0	1	0	NA	RB	LOAD 8 bits in next address into RB

In this command set, the CPU interacts with both registers and memory. The command DATA is different with all commands we see before, it has 2 bytes. The first one is the command, the second one is the data for the command. See example below for more details.

Example 1: IR = 0b00000111

- This is LOAD command
- RA is 1
- RB is 3

CPU takes the data in R1 as the address pointer, then it loads the data of that address into register R3. See image below.

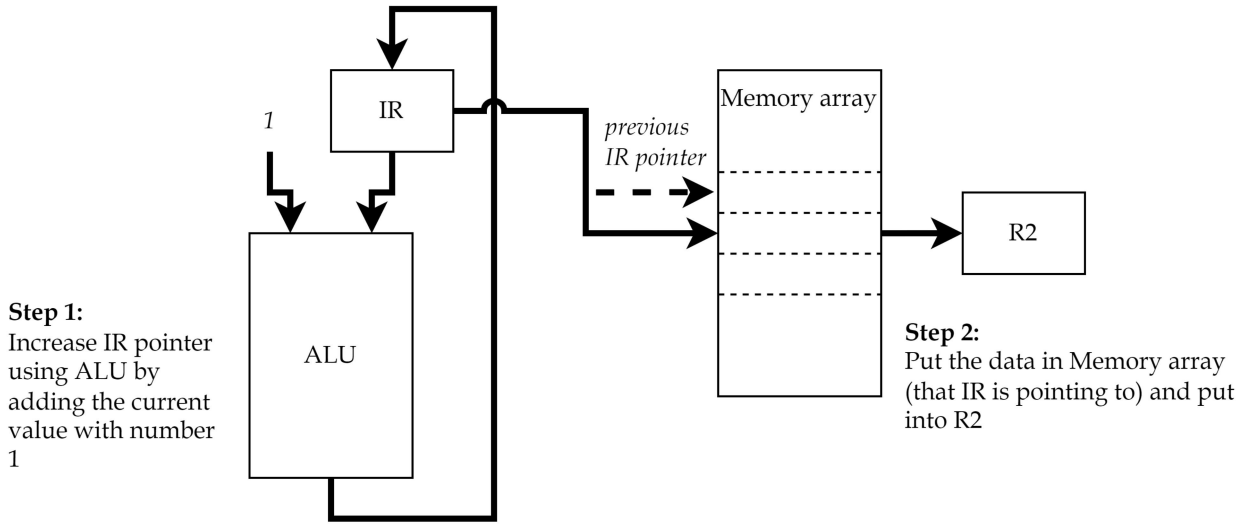


Note that in this command, the IR pointer doesn't change, so CPU will go to the next address of the IR pointer to operate next command. The IR pointer's behavior is different with the command DATA in the next example.

Example 2: IR = 0b00100010

- This is DATA command
- RA is don't care, the value is ignored
- RB is 2

CPU points to the next address of current pointer, then it loads the data of that address into register R2. In this command, there are 2 locations of memory join into the command instead of 1 as other command. So, CPU will jump to the next 2 adress (from the original pointer) to operate next command. See image below.



Note that the use of ALU in step 1 is just a design of choice. (Which means there maybe other better solution in the world)

Jump

In normal operation, IR automatically increase the value to the next one, after the current command has been processed. This command set is used to jump the IR pointer around the memory array. Except JMPR, all other instruction store the adress that need to jump to in the next byte.

Command	Length	7	6	5	4	3:2	1:0	Description
JMPR	1	0	0	1	1	NA	RB	JUMP TO address in RB
JMP	2	0	1	0	0	NA	NA	JUMP TO address in next byte
JZ	2	0	1	0	1	00	01	JUMP IF answer is zero
JE	2	0	1	0	1	00	10	JUMP IF A equals B
JA	2	0	1	0	1	01	00	JUMP IF A is larger than B
JC	2	0	1	0	1	10	00	JUMP IF CARRY is on
JCA	2	0	1	0	1	11	00	JUMP IF CARRY or A larger
JCE	2	0	1	0	1	10	10	JUMP IF CARRY or A equal B
JCZ	2	0	1	0	1	10	01	JUMP IF CARRY or answer is zero
JAE	2	0	1	0	1	01	10	JUMP IF A is larger or equal to B
JAZ	2	0	1	0	1	01	01	JUMP IF A is larger or answer is zero
JEZ	2	0	1	0	1	00	11	JUMP IF A equals B or answer is zero
JCAE	2	0	1	0	1	11	10	JUMP IF CARRY or A larger or equal to B

Command	Length	7	6	5	4	3:2	1:0	Description
JCAZ	2	0	1	0	1	11	01	JUMP IF CARRY or A larger or zero
JCEZ	2	0	1	0	1	10	11	JUMP IF CARRY or A equals B or zero
JAEZ	2	0	1	0	1	01	11	JUMP IF A larger or equal to B or zero
JCAEZ	2	0	1	0	1	11	11	JUMP IF CARRY or A larger or equal or zero

Most of commands in this set use the 4 LSB to distiguish with each other rather than point to register as in previous command sets.

In the above table, we can see each bit in the range [3:0] of the instruction enables 1 flag to compare. See table below.

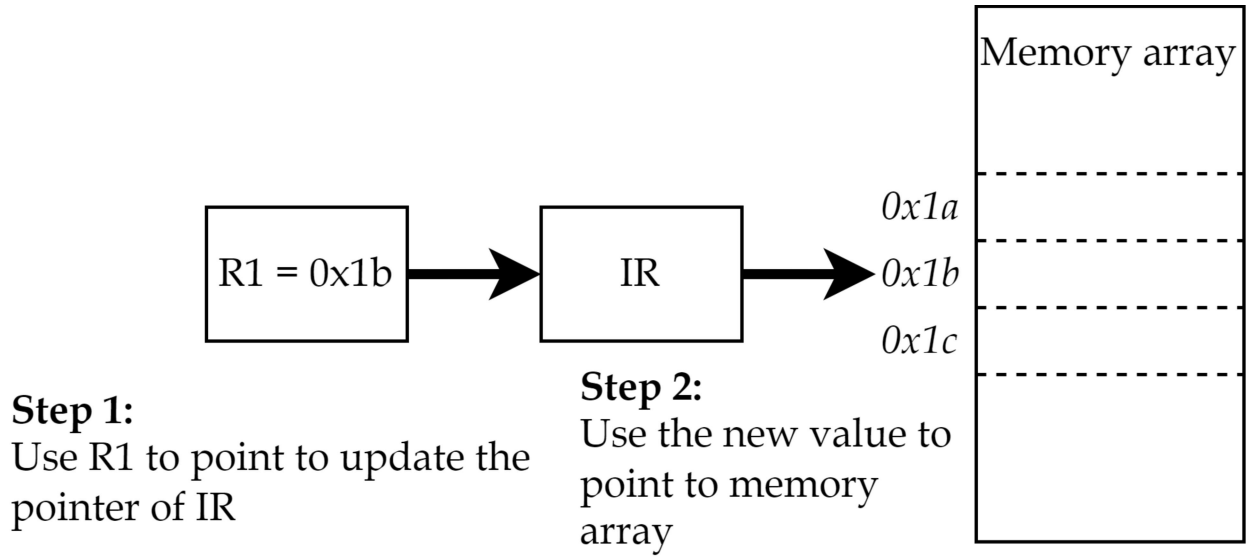
IDX	Flag
3	Carry
2	Larger
1	Equal
0	Zero

Future flag comparison design can base on this to simplify the code.

Example 1: IR = 0b00110001

- This is JUMPR command
- RA is ignored; thus, the value is not ignored.
- RB is 1

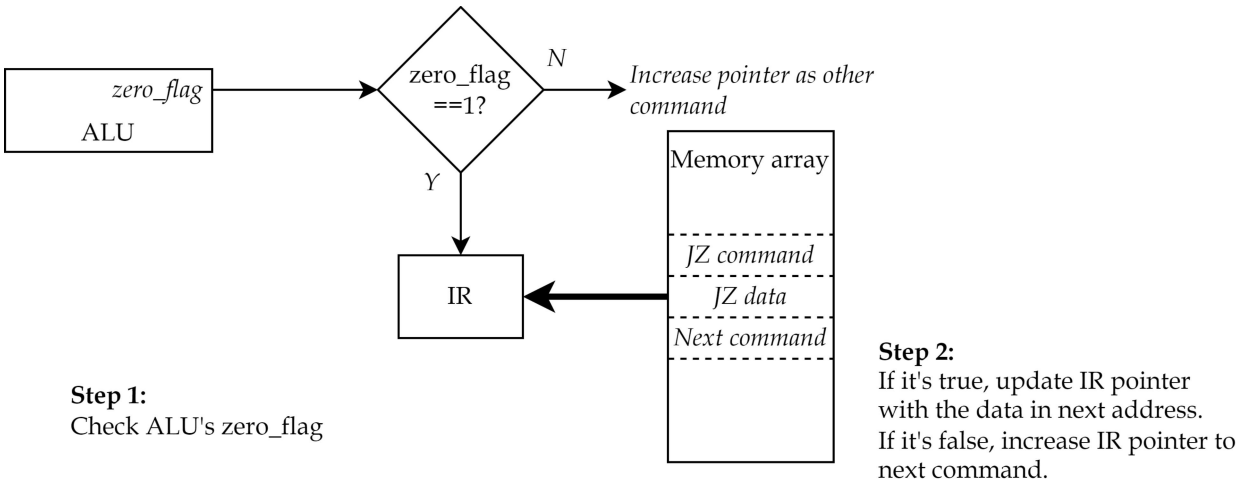
CPU load the value of R1 to get the address to jump, then it points the IR pointer to the loaded address in the memory and process next command from there. See image below.



Example 2: IR = 0b001010001

- This is JZ command

CPU checks the flags from ALU block, if the ZERO flag is set (1), it will jump the IR pointer to the memory address that stored in the next address. Otherwise, if skip the next address and continue from the next 2 address. See image below.



This (and other similar commands) command uses the ALU flags to decide operation. The ALU flags is set from ALU command and only be cleared by CLF command.

Miscellaneous

Command	Length	7	6	5	4	3:2	1:0	Description
CLF	1	0	1	1	0	NA	NA	CLEAR ALL FLAGS
END	1	1	1	0	0	11	11	END

CLF is used to clear all flags in ALU. END is used to tell the CPU the end of program and it stops operating.

Important Note: The 4 MSB of END command is the same as ALU AND (in this case it is AND the register R3 with itself), thus it should be a confliction. However, we still can use this code for END because it is meaningless when you want to operate an AND operation of the same value, since it doesn't change anything. Thus, if CPU see exactly the code, it should understand this is the END command instead of ALU AND.