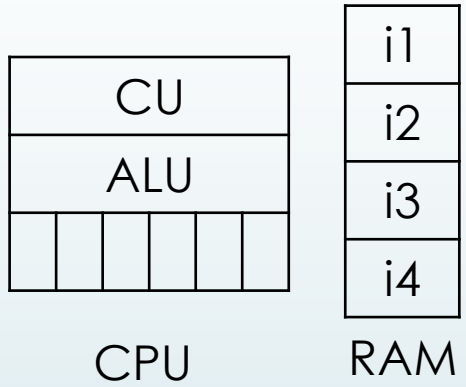
Instruction Set Architecture (ISA)

A computer program is a set of instructions.



Information that must be present in an instruction.

- Operation: Add, sub, etc.
- Where to get the data: memory address or register
- Where to put the result: memory address or register
- Where to find the next instruction: memory address

d=(a+b)*c

2	7	3	6
a	b	c	d

Instruction parts

An instruction is composed of:

Opcode	Operands

Example:

Add	[100]	[104]

Add values in [100] and [104] then store the result in [100], where [100] and [104] are two memory locations.

Instruction formats (1)

Formats used to represent an instruction:

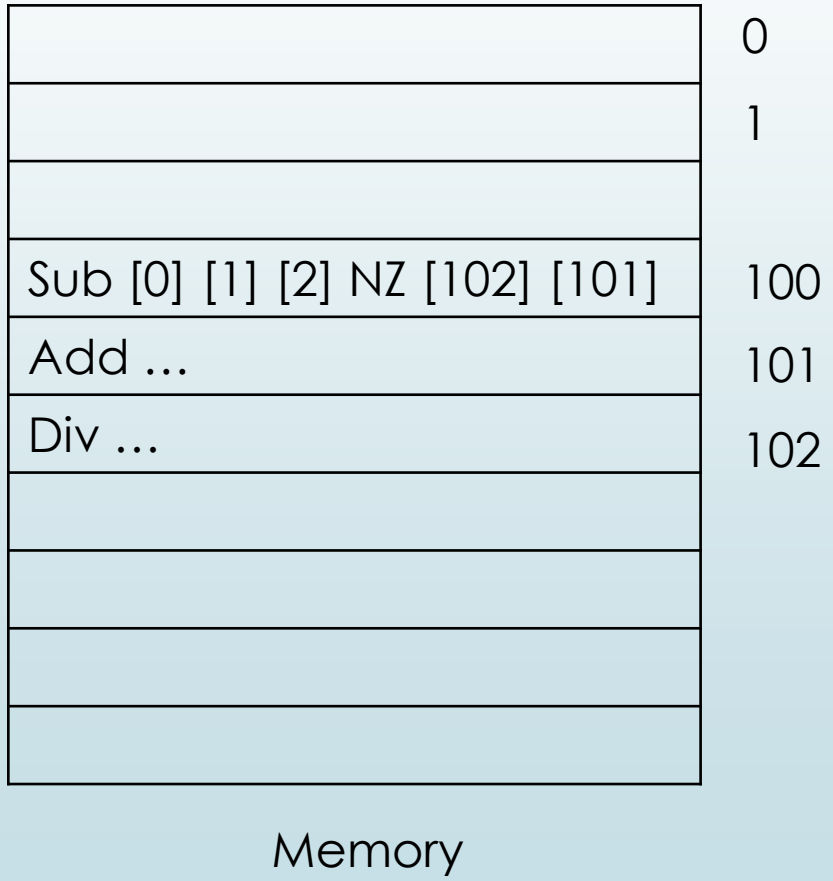
a) Machine with a single type of instruction

Opcode	Destination	Source1	Source2	Condition	Next/ True	Next/ False
Sub	[0]	[1]	[2]	NZ	[102]	[101]

Subtract the value stored in location 2 from that in location 1and then store the result in location 0. If the result is not zero then get the next instruction from location 102, otherwise get it from location 101.

Disadvantage: long instruction (for example:10 bits for the opcode, 6 bits for the condition, 16 bits for each address, then the instruction would have to be 96 bits long).

Instruction formats (1)



Instruction formats (2)

b) Three address machine: not all instructions need to change the order of instruction execution.

We need a register PC (Program Counter) to store the address of the next instruction.

We split the previous instruction into two types:

ALU instruction

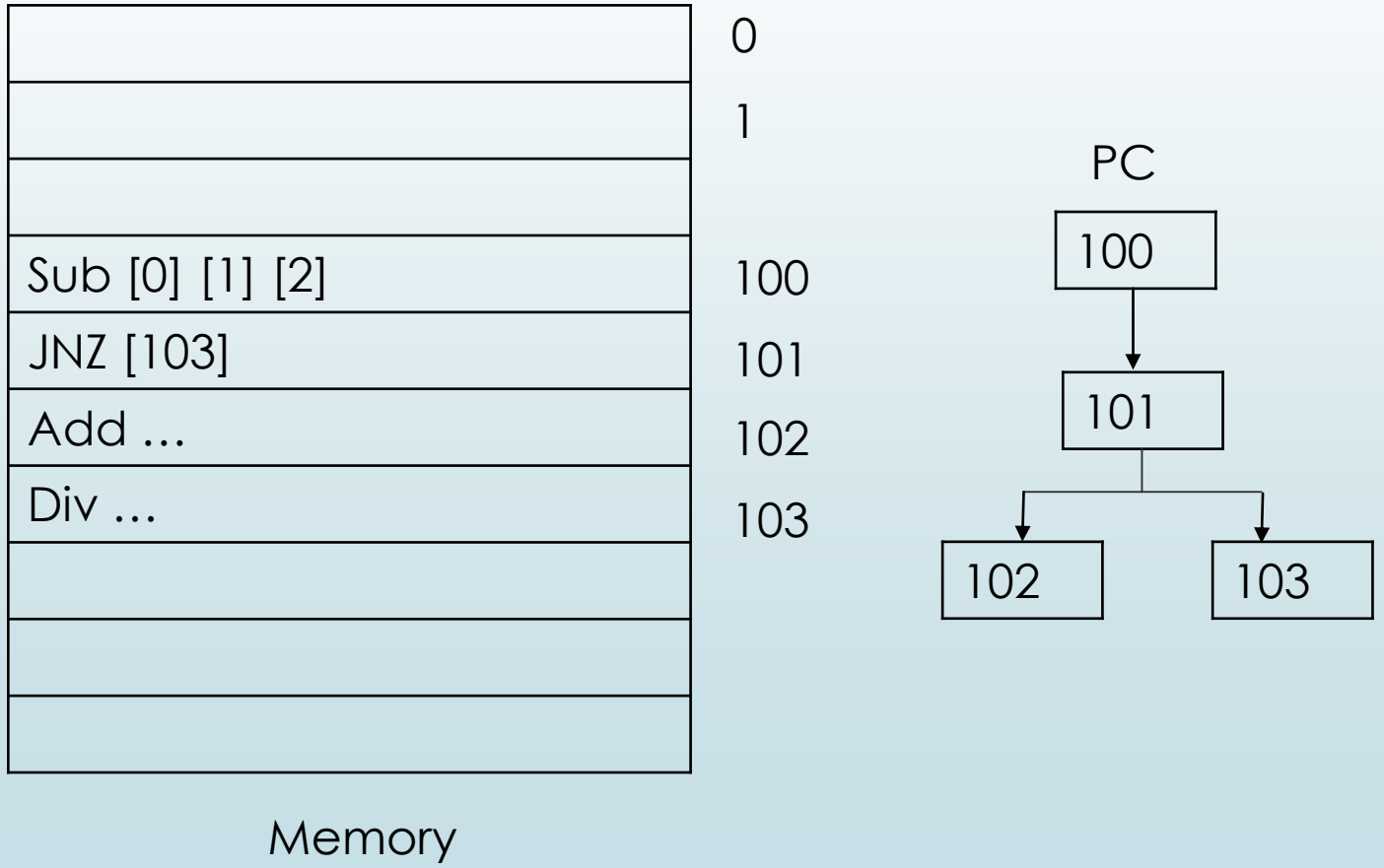
Opcode	Destination	Source1	Source2
Sub	[0]	[1]	[2]

Control instruction

Opcode	Next/True
JNZ	[103]

JNZ: Jump Not Zero

Instruction formats (2)



Instruction formats (2)

Advantage:

Reduce memory.

Disadvantage:

Two instructions to be executed but such operations are rare.

Instruction formats (3)

c) Two address machine: define a Mov operation (copy only). It is useful in case of a=b.

ALU instruction

Opcode	Destination	Source
Mov	[0]	[1]
SUB	[0]	[2]

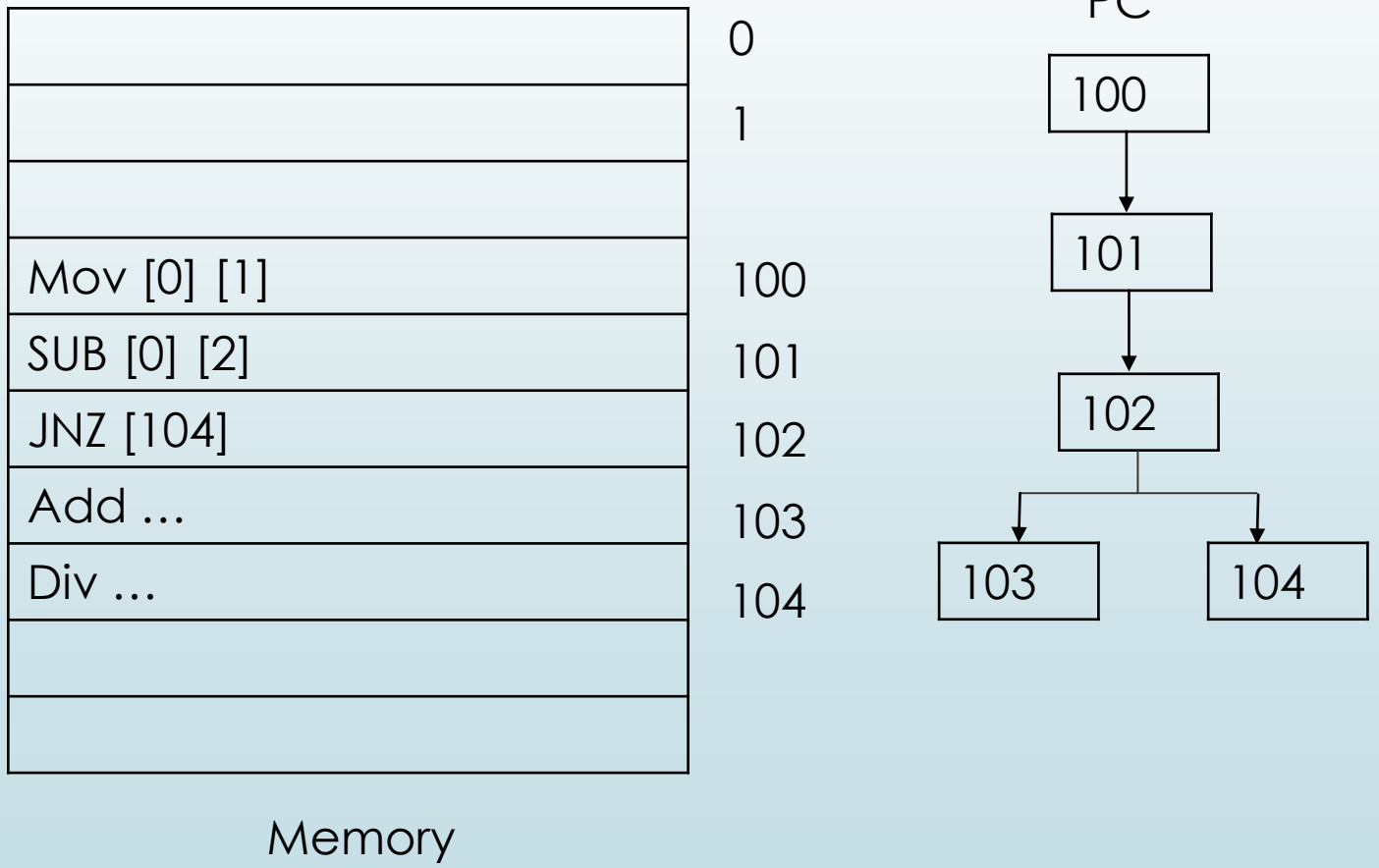
Control instruction

Opcode	Next/True
JNZ	[104]

Sub Destination Source ≡ Sub Destination Destination Source

2-address 3-address

Instruction formats (3)



Instruction formats (4)

d) One address machine:

- Use the register AC (accumulator).
- Define the operations load and store (Copy only).

ALU instruction

Opcode	Source
Load	[1]
Sub	[2]
Store	[0]

AC ← M[1]

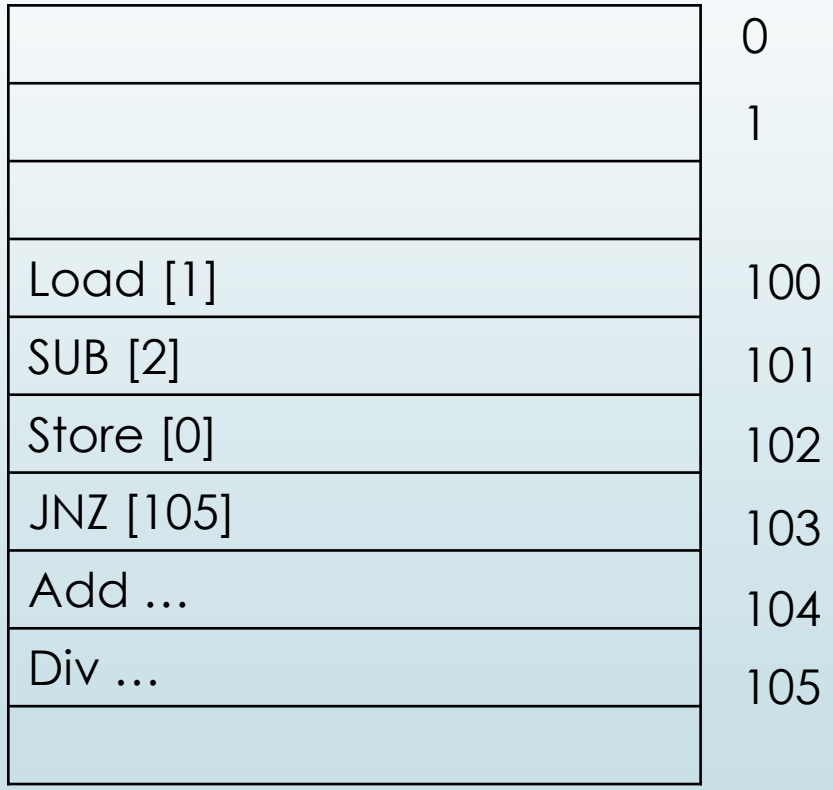
AC ←AC - M[2]

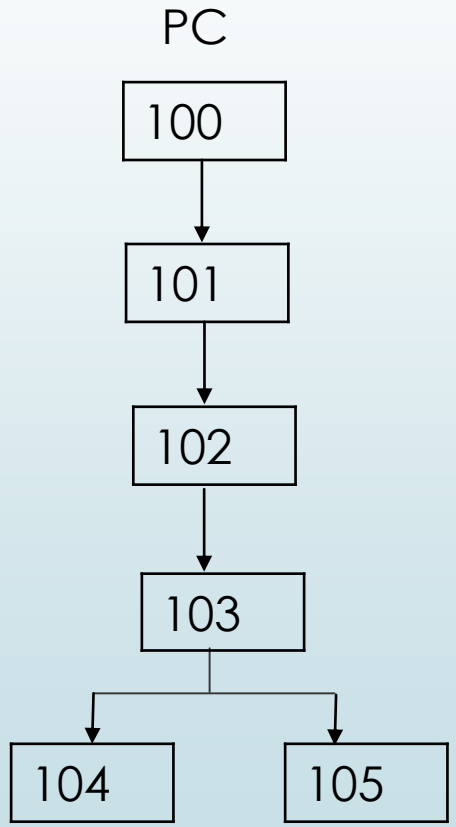
M[0] ←AC

Control instruction

Opcode	Next/True
JNZ	[105]

Instruction formats (4)





Instruction formats (5)

e) Zero address or stack machine: use Push and Pop operations.

Stack instruction

Opcode	Source
Push	[2]
Push	[1]
Pop	[0]

ALU instruction

Opcode
Sub

Control instruction

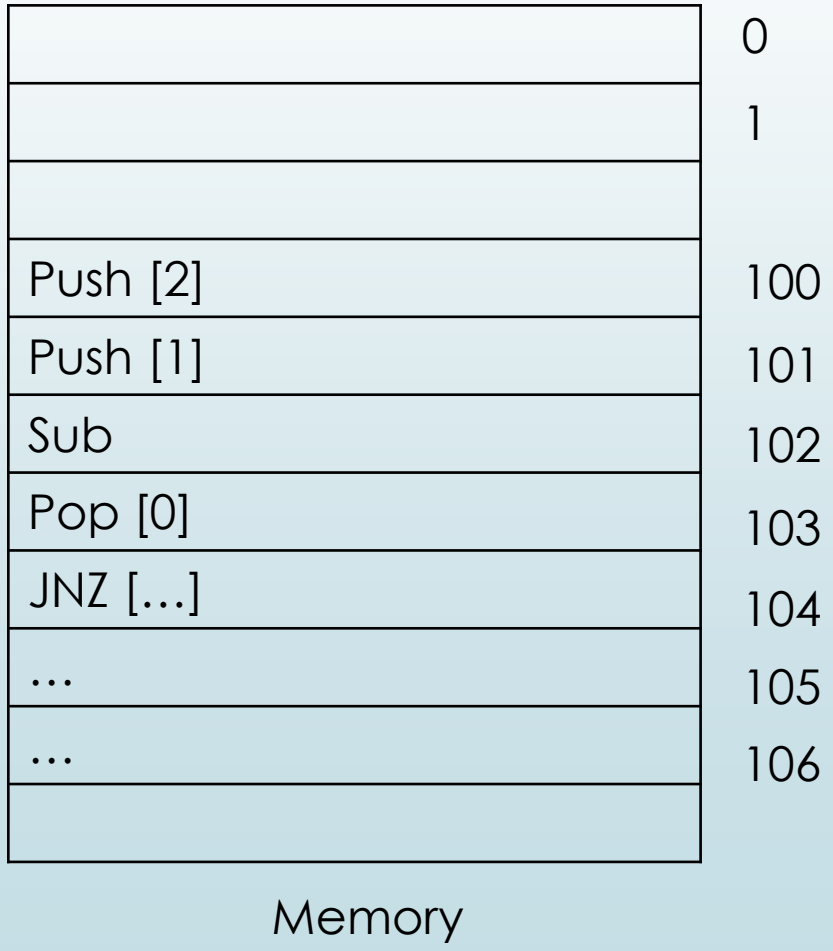
Opcode	Next/True
JNZ	[105]

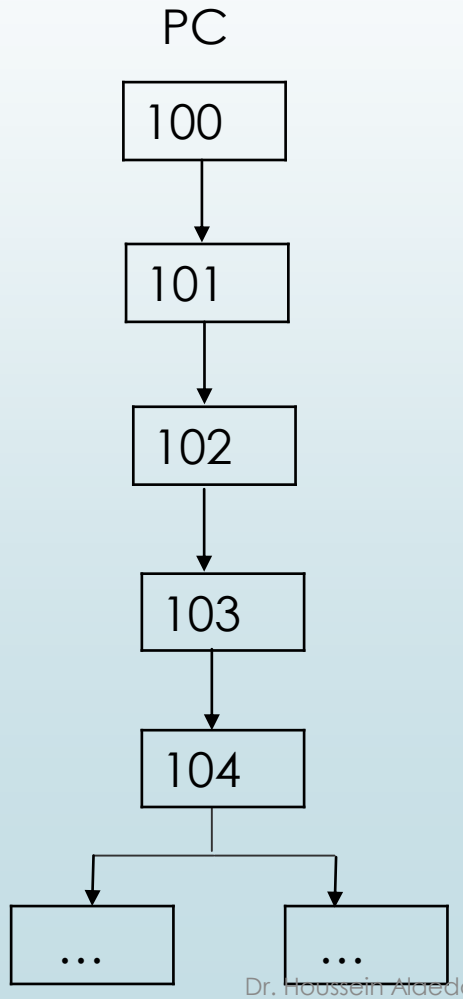
Push [1]: move M[1] to the top of stack.

Pop [0]: move the top of stack to M[0]

Sub: pop the two top numbers from the stack, sub the numbers, and push the result into the stack.

Instruction formats (5)





Exercise

Convert the following code into 0 address machine.

```
X=Y-Z;
if(X==0)
   X=Y*Z;
X=Y/Z;
```

Where X, Y and Z are in locations 0, 1 and 2 respectively.

Solution

```
X=Y-Z
if(X==0)
   X=Y*Z;
X=Y/Z;
```

Value of X	0
Value of Y	1
Value of Z	2
Push [2]	100
Push [1]	101
Sub	102
Pop[0]	103
JNZ[109]	104
Push[1]	105
Push[2]	106
Mul	107
Pop[0]	108
Push[2]	109
Push[1]	110
Div	111
Pop[0]	112

Instruction formats (6)

f) Load/Store machine:

- Only load and store operations can access memory
- All other operations use registers as operands.

Later …

(MIPS language)

Exercise

Consider the following C-code:

```
X=Y*(Y+Z);
```

Where X, Y and Z are in locations 0, 1 and 2 respectively.

Convert this code into 3, 2, 1, 0 address machines.