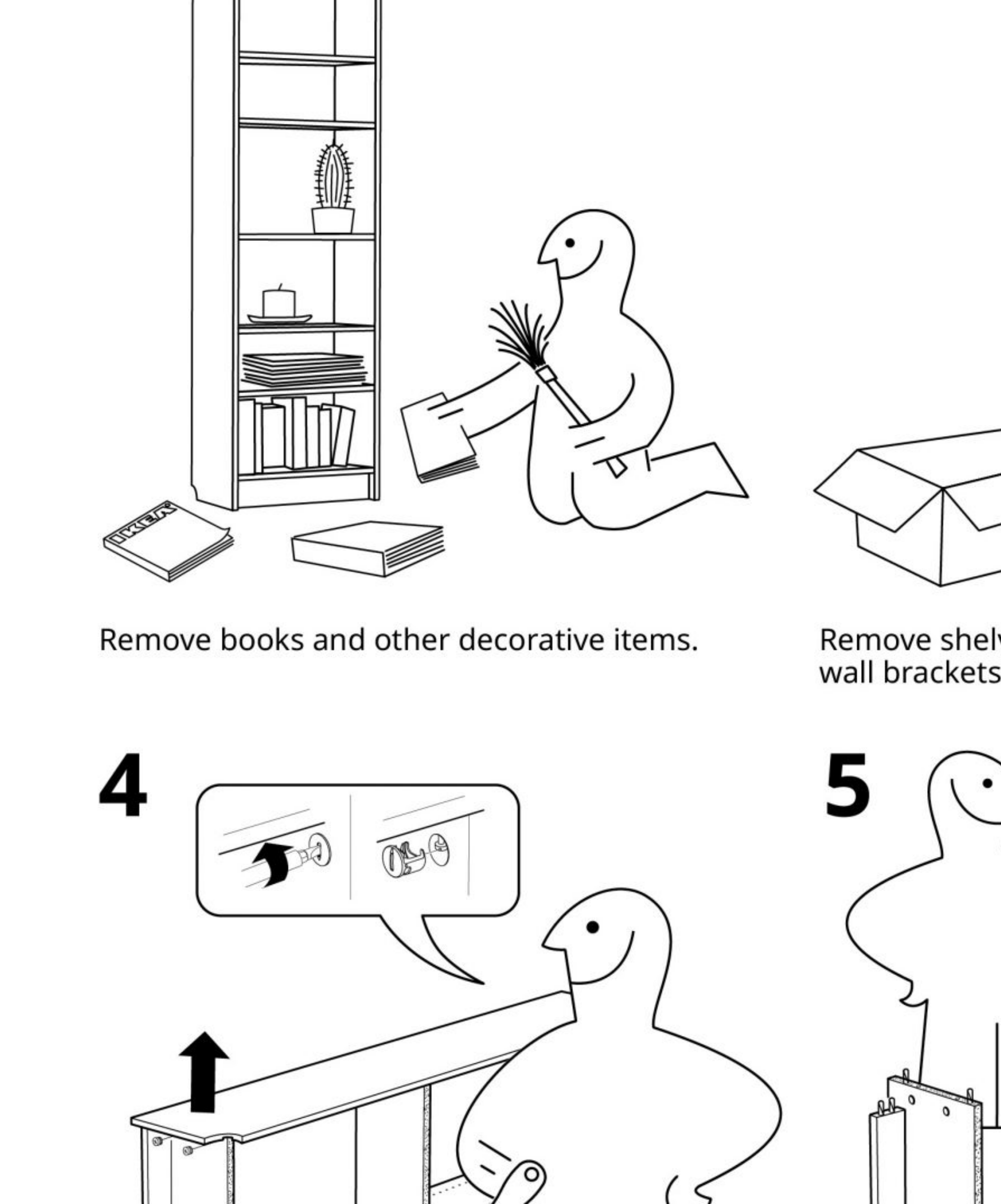
DR EDWARD ANSTEAD

LOGIC AND COMPUTER ARCHITECTURE

Overview of a simple computer
and Instruction set architecture



TODAY...

- **LO2:** Explain the architecture of standard computer hardware and how it is used to represent and calculate with data
- We will look at the principles of building a example computer system and its components, this will be followed by a description of its instruction set.
- **LO3:** Explain the methods used to represent a wide variety of types of data on a computer
- As we uncover the principles of a computer instruction set we will see its application in writing simple programs.

TOPICS

- Further work with the MARIE example computer
- additional instructions
- Microoperations and Register Transfer Language

MARIE EXAMPLE PROGRAM: CALCULATOR

- Multiply two values given as input
- We are going to make use of…
  - variables
  - skipcond
  - jump
  - labels

MARIE EXAMPLE PROGRAM: CALCULATOR

- Multiply two values given as input
- We are going to make use of…
  - variables
  - skipcond
  - jump
  - labels

A PROGRAMMER FRIENDLY LABEL IN THE CODE THAT IDENTIFIES A PARTICULAR LINE

SKIPCOND

`SKIPCOND` skips the next instruction according to the value of the AC.

```
If IR[11 - 10] = 00 then
     If AC < 0 then PC ← PC + 1
else If IR[11 - 10] = 01 then
     If AC = 0 then PC ← PC + 1
else If IR[11 - 10] = 10 then
     If AC > 0 then PC ← PC + 1
```

SKIPCOND

`SKIPCOND` skips the next instruction according to the value of the AC.

```
If IR[11 - 10] = 00 then
     If AC < 0 then PC ← PC + 1
else If IR[11 - 10] = 01 then
     If AC = 0 then PC ← PC + 1
else If IR[11 - 10] = 10 then
     If AC > 0 then PC ← PC + 1
```

IR 000(HEX) AC < 0
IR 400(HEX) AC = 0
IR 800(HEX) AC > 0

SKIPCOND

`SKIPCOND` skips the next instruction according to the value of the AC.

```
skipcond 000 /if the value in the AC is <0 skip next line
skipcond 400 /if the value in the AC ==0 skip next line
skipcond 800 /if the value in the AC is >0 skip next line
```

SKIPCOND

SKIPCOND skips the next instruction according to the value of the AC.

```
skipcond 000 /if the value in the AC is <0 skip next line
skipcond 400 /if the value in the AC ==0 skip next line
skipcond 800 /if the value in the AC is >0 skip next line
```

SKIP NEXT LINE - ADD 1 TO PC

JUMP

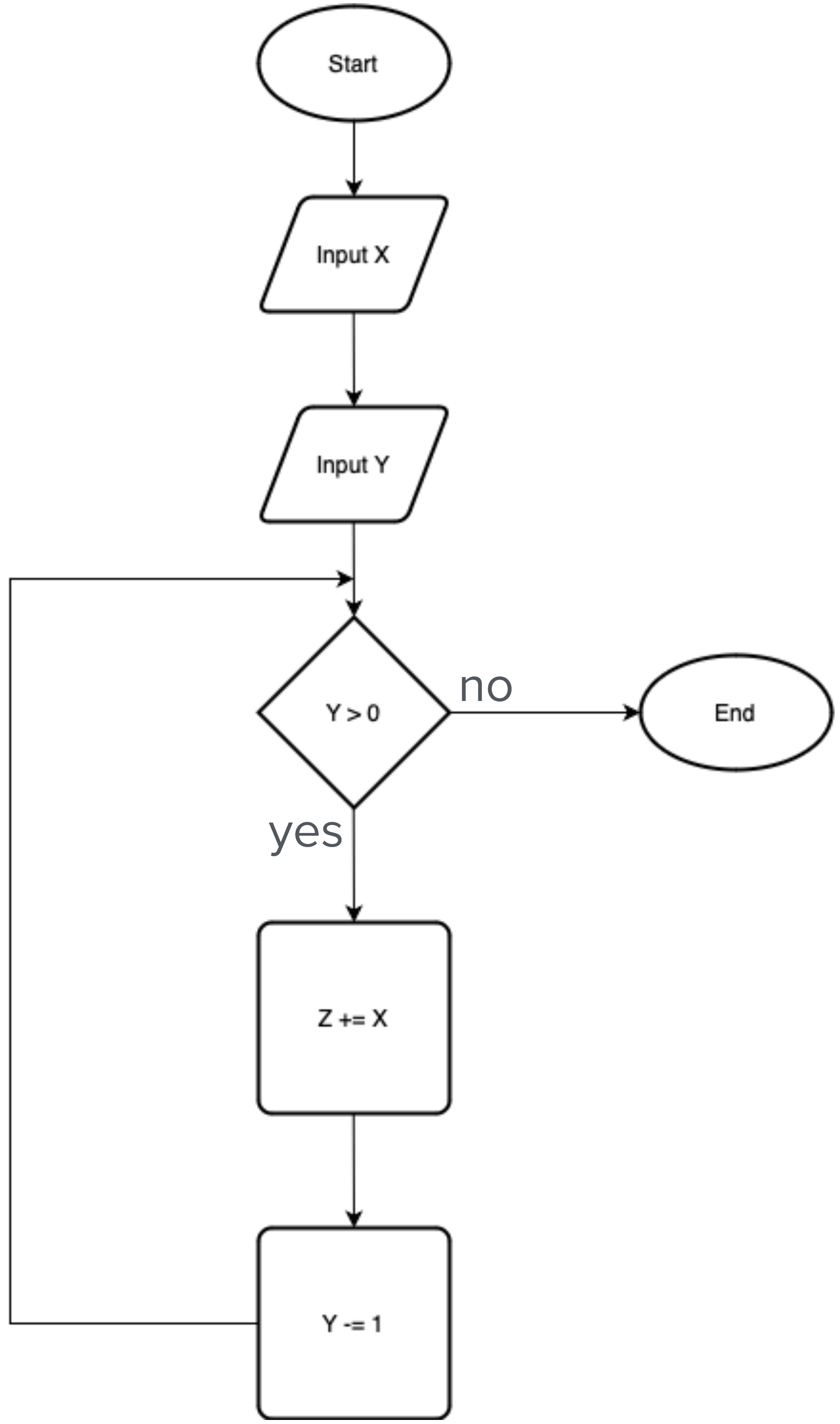
- jump to an instruction at the specified address
- We can make this much more readable with labels!

```
jump 0A1 /Jump to instruction at 0A1
jump LAB /Jump to the instruction with the label LAB
```

- By Combining jump and skipcond we can make the equivalent constructs of the loops and conditionals we are familiar with in high level programming

PROGRAM FLOWCHART

- Multiply two values given as input
- We are going to make use of…
  - variables
  - skipcond
  - jump
  - labels



EXAMPLE PROGRAM IF/ELSE

- write a program that implements the following pseudocode:

```
if X = Y then
 X = X x 2
else
 Y = Y - X
```

EXTENDING OUR INSTRUCTION SET

MARIE Instructions

Binary	Hex	Instructio	Meaning
0001	1	Load X	Load contents of address X into AC
0010	2	Store X	Store contents of AC in address X
0011	3	Add X	Add the contents of address X to AC
0100	4	Subt X	Subtract the contents of address X to AC
0101	5	Input	Input a value from the keyboard into AC
0110	6	Output	Output the value from the AC to the display
0111	7	Halt	Terminate the program
1000	8	Skipcond	Skip next instruction on condition
1001	9	Jump X	Load the value of X into the PC

MARIE Instructions

Binary	Hex	Instruction	Meaning
0000	**0**	**JnS X**	**Store the PC at address X and jump to X+1**
0001	1	Load X	Load contents of address X into AC
0010	2	Store X	Store contents of AC in address X
0011	3	Add X	Add the contents of address X to AC
0100	4	Subt X	Subtract the contents of address X to AC
0101	5	Input	Input a value from the keyboard into AC
0110	6	Output	Output the value from the AC to the display
0111	7	Halt	Terminate the program
1000	8	Skipcond	Skip next instruction on condition
1001	9	Jump X	Load the value of X into the PC
1010	**A**	**Clear**	**Put all zeros in the AC**
1011	**B**	**AddI X**	**Add Indirect: Go to address X. Use the value at X as the actual address of the data operand to add to the AC**
1100	**C**	**JumpI X**	**Jump Indirect: Go to address X. Use the value at X as the actual address of the location to jump to.**
1101	**D**	**LoadI X**	**Load indirect: Go to address X. Use the value at X as the actual address of the operand to load into the AC.**
1110	**E**	**StoreI X**	**Store Indirect: Go to the address X. Use the value at X as the destination address for storing the value in the accumulator.**

EXAMPLE PROGRAM: NUMBER TOTALISER

- Add together a series of numbers stored in the machines memory
- To do this we will use
  - variables
  - labels
  - add indirect
  - Jump
  - skipcond

EXAMPLE PROGRAM: STRING OUTPUT

- Output a string from the computers memory.
- To do this we will use
  - variables
  - labels
  - load indirect
  - Jump
  - skipcond

MICRO INSTRUCTIONS

- We have seen the operations of a simple computer simulator using machine code instructions.
  - Each of these instructions is built from a series of smaller units.
    - **micro-instructions**
  - Understanding this will help to conceptualise the operation of the control unit.
  - We can see the micro-instructions for MARIE from the simulator

Program execution

Figure adapted from Stallings 2022 Chapter 19

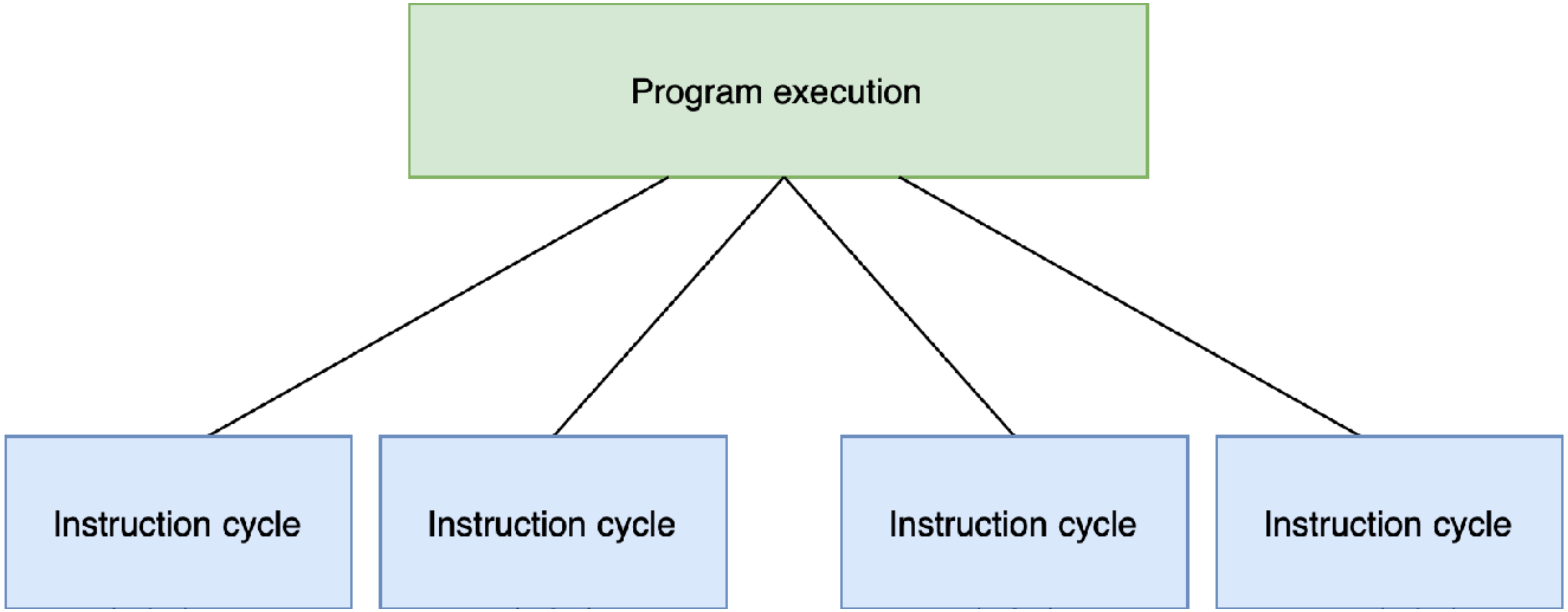


Figure adapted from Stallings 2022 Chapter 19

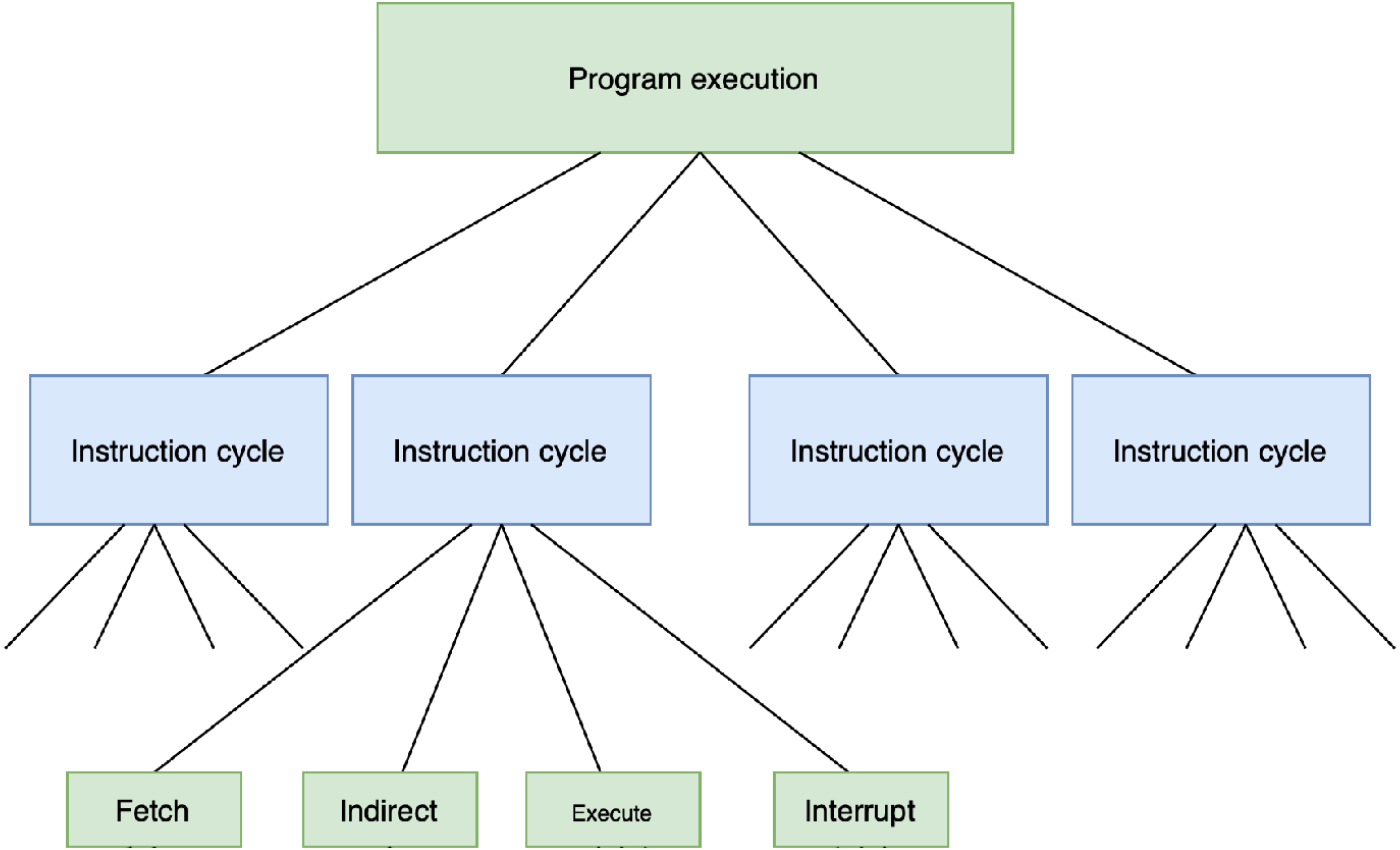


Figure adapted from Stallings 2022 Chapter 19

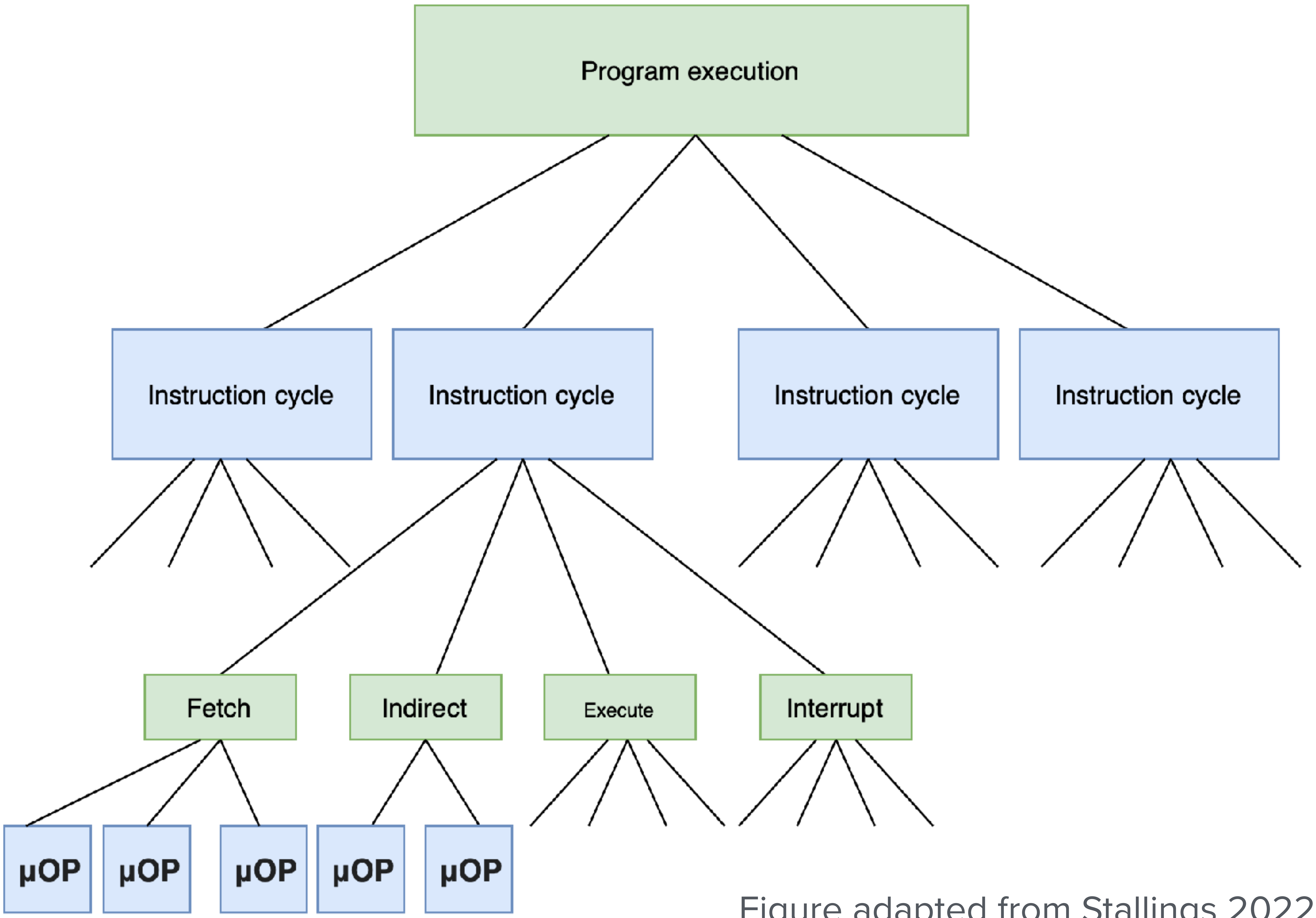


Figure adapted from Stallings 2022 Chapter 19

REGISTER TRANSFER LANGUAGE / NOTATION

- Micro operations are represented in register transfer language (sometimes register transfer notation)
- RTL shows the transfer of data between registers and memory in an operation

RTL	Description
MAR ← PC	load address from PC into MAR
MBR ← M[MAR]	Load data at memory location stored in MAR into MBR
IR ← MBR	Load data from MBR into IR
PC ← PC + 1	set value of PC to be PC + 1

FETCH CYCLE

- The beginning phase of an instructions execution
- During the fetch cycle an instruction is read from memory
- In MARIE four registers are involved in the fetch cycle.
  - Memory address register (MAR)
  - Memory buffer register (MBR)
  - Program counter (PC)
  - Instruction register (IR)
- The fetch cycle is the same for all operations

FETCH CYCLE

Time unit	RTL
t1	MAR ← PC
t2	MBR ← M[MAR]
t3	PC ← PC+1
t4	IR ← MBR

FETCH CYCLE

Time unit	RTL
t1	MAR ← PC
t2	MBR ← M[MAR]
t3	PC ← PC+1
t4	IR ← MBR

Register	Data
MAR	
MBR	
PC	0000 0000 1010
IR	
AC	
Input	
Output	

FETCH CYCLE

Time unit	RTL
t1	**MAR ← PC**
t2	MBR ← M[MAR]
t3	PC ← PC+1
t4	IR ← MBR

Register	Data
MAR	0000 0000 1010
MBR	
PC	0000 0000 1010
IR	
AC	
Input	
Output	

FETCH CYCLE

Time unit	RTL
t1	MAR ← PC
t2	**MBR ← M[MAR]**
t3	PC ← PC+1
t4	IR ← MBR

Register	Data
MAR	0000 0000 1010
MBR	0010 0000 0000 1100
PC	0000 0000 1010
IR	
AC	
Input	
Output	

FETCH CYCLE

Time unit	RTL
t1	MAR ← PC
t2	MBR ← M[MAR]
t3	**PC ← PC+1**
t4	IR ← MBR

Register	**Data**
MAR	0000 0000 1010
MBR	0010 0000 0000 1100
PC	0000 0000 1011
IR	
AC	
Input	
Output	

FETCH CYCLE

Time unit	RTL
t1	MAR ← PC
t2	MBR ← M[MAR]
t3	PC ← PC+1
t4	**IR ← MBR**

Register	**Data**
MAR	0000 0000 1010
MBR	0010 0000 0000 1100
PC	0000 0000 1011
IR	0010 0000 0000 1100
AC	
Input	
Output	

INDIRECT CYCLE

- After fetching The operands of the instruction need obtaining from memory
- If this is indirectly addressed an extra step is required.

INDIRECT CYCLE

Time unit	RTL
t1	MAR ← IR[Address]
t2	MBR ← M[MAR]
t3	MAR ← MBR[Address]

Register	Data
MAR	0000 0000 1010
MBR	1101 0000 0000 1100
PC	0000 0000 1011
IR	1101 0000 0000 1100
AC	
Input	
Output	

INDIRECT CYCLE

Time unit	RTL
t1	**MAR ← IR[Address]**
t2	MBR ← M[MAR]
t3	MAR ← MBR[Address]

Register	**Data**
MAR	0000 0000 1100
MBR	1101 0000 0000 1100
PC	0000 0000 1011
IR	1101 0000 0000 1100
AC	
Input	
Output	

INDIRECT CYCLE

Time unit	RTL
t1	MAR ← IR[Address]
t2	**MBR ← M[MAR]**
t3	MAR ← MBR[Address]

Register	Data
MAR	0000 0000 1100
MBR	0000 0000 0010 1011
PC	0000 0000 1011
IR	1101 0000 0000 1100
AC	
Input	
Output	

INDIRECT CYCLE

Time unit	RTL
t1	MAR ← IR[Address]
t2	MBR ← M[MAR]
t3	**MAR ← MBR[Address]**

Register	**Data**
MAR	0000 0010 1011
MBR	0000 0000 0010 1011
PC	0000 0000 1011
IR	1101 0000 0000 1100
AC	
Input	
Output	

INDIRECT CYCLE - VARIATION (AS DETAILED IN STALLINGS 2022)

Time unit	RTL
t1	MAR ← IR[Address]
t2	MBR ← M[MAR]
t3	**IR(Address) ← MBR[Address]**

Register	**Data**
MAR	0000 0000 1100
MBR	0000 0000 0010 1011
PC	0000 0000 1011
IR	1101 **0000 0010 1011**
AC	
Input	
Output	

EXECUTE AND INTERRUPT CYCLES

- Execute Cycle
  - Each opcode will have unique micro-operations for execution. We can see these in the RTL output in the MARIE simulator.
- Interrupt Cycle
  - MARIE doesn't support interrupts
  - A simple process might involve:

t1: MBR ← PC
t2: MAR ← [A free memory address]
t3: PC ← [Interrupt routine address]
t4: M ← (MBR)

MICRO-OPERATION SUMMARY

- Micro-operations perform one of the following
  - Transfer data between registers
  - transfer data from a register to an external interface (system bus)
  - transfer data from an external interface to a register
  - perform an arithmetic or logical operation using registers