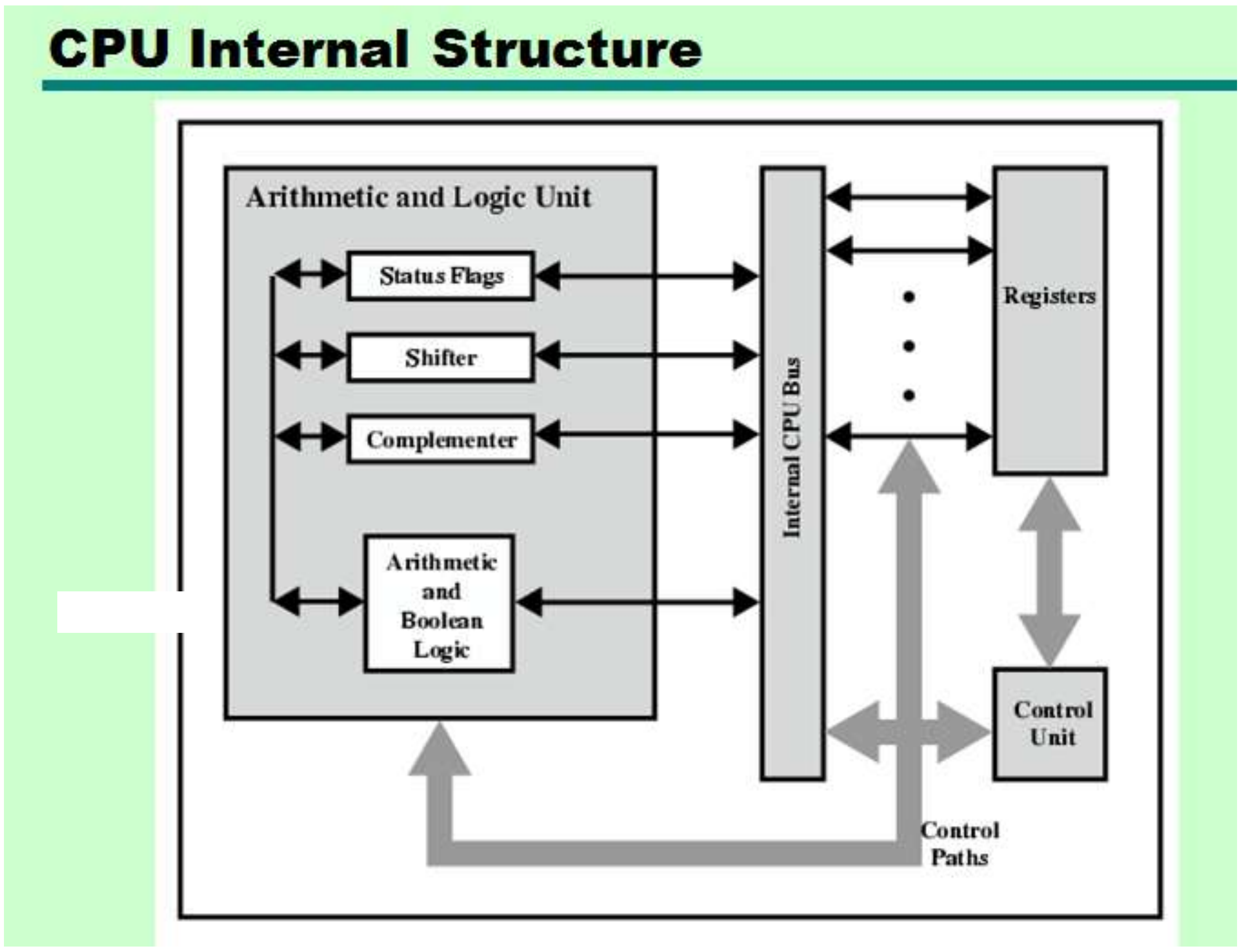
Part (6)

CPU BASICS

A typical CPU has three major components:
1- register set,
2- arithmetic logic unit (ALU),
3- control unit (CU).
The figure below shows the internal structure of the CPU .
The CPU fetches instructions from memory, reads and writes data from and to memory, and transfers data from and to input/output devices.



Instruction execution cycle:
1- fetch: fetching next instruction(using PC) from memory into IR.
2-decode:decoding the instruction.
3- execute: executing the instruction.

REGISTER SET:

• The register set differs from one computer architecture to another. It is usually a combination of general-purpose and special purpose registers

• *General-purpose registers* can be used for multiple purposes and assigned to a variety of functions by the programmer.

• *Special-purpose registers* are restricted to only specific functions.

Memory Access Registers:

Two registers are essential in memory write and read operations: the memory data register (**MDR**) and memory address register (**MAR**). The MDR and MAR are used exclusively by the CPU and are not directly accessible to programmers.

In order to perform a **write** operation into a specified memory location, the MDR and MAR are used as follows:
1. The word to be stored into the memory location is first loaded by the CPU into **MDR**.
2. The address of the location into which the word is to be stored is loaded by the CPU into a **MAR**.
3. A write signal is issued by the CPU.

Similarly, to perform a memory **read** operation, the MDR and MAR are used as follows:
1. The address of the location from which the word is to be read is loaded into the **MAR.**
2. A read signal is issued by the CPU.
3. The required word will be loaded by the memory into the **MDR** ready for use by the CPU.

Instruction Fetching Registers:
Two main registers are involved in fetching an instruction for execution: the program counter (**PC**) and the instruction register (**IR**). The PC is the register that contains the address of the next instruction to be fetched. The fetched instruction is loaded in the IR .

Condition Register: Program status word (**PSW**) register contains bits that are set by the CPU to indicate the current status of an executing program. These indicators are typically for arithmetic operations, interrupts,…etc.

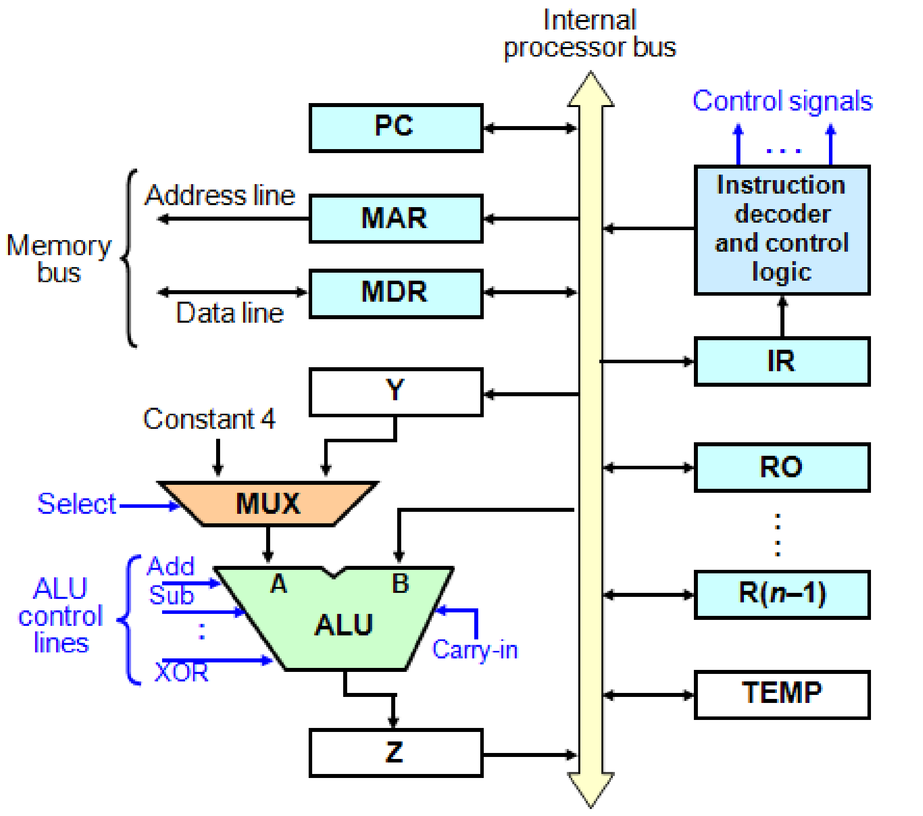
Special-Purpose Address Registers:

• In index addressing, the **index register** holds an address displacement which when added to a constant, the address of the operand is obtained.

• A **segment register** holds the address of the base of the segment.

• A specific register, called the **stack pointer (SP**), is used to indicate the stack location that can be addressed. In the stack push operation, the SP is incremented and in pop operation the SP is decremented.

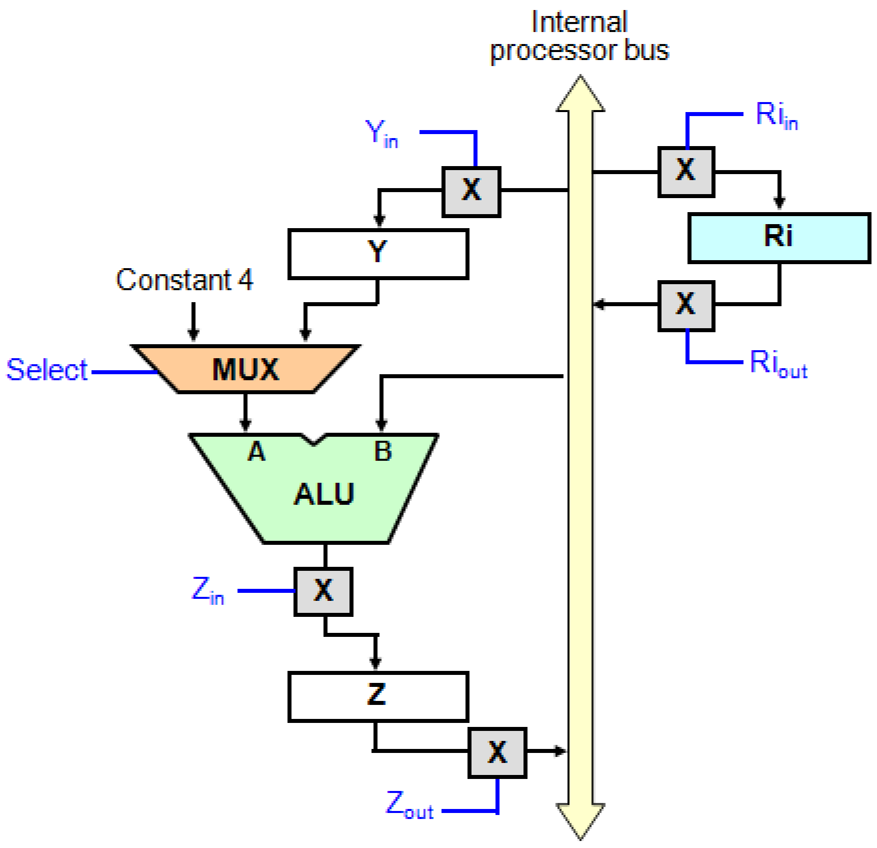
Single-bus Organization



Register transfer:

For each register Ri, there are two control signals:
Ri_{in} used to load the data on the bus into the register.
Ri_{out} used to place the register's contents on the bus.

EX: R1 $\rightarrow$ R4 will be
$R1_{out}$, $R4_{in}$



To perform arithmetic/logic operation:

.The ALU is a combinational circuit that has no internal storage.
.ALU gets the two operands from MUX and bus. The result is temporarily stored in register Z.

EX: What is the sequence of operation to the following instruction:
ADD R1 , R2
1. $R1_{out}$, Y_{in}
2.$R2_{out}$,Select Y, Add, Z_{in}
3.Z_{out} ,$R2_{in}$

Fetching aword from memory:

EX: MOVE (R1) ,R2

1. $R1_{out}$,MAR_{in} , Read
2. WMF(wait to memory function complete)
3. MDR_{out} ,$R2_{in}$

Execution of a complete instruction:

EX: Write the control steps to fetch and execute the following instruction:

ADD (R3), R1

Note: $PC_{new} = PC_{old}$ + constant 4

<u>Execution of Branch instructions:</u>

A branch instruction replaces the content of PC with the:

branch target address(PC_{new})= offset **X** + PC_{old}
offset **X** (given in the branch instruction).

The required control steps to fetch and execute unconditional branch as follows:

1 PC_{out}, MAR_{in}, Read, Select 4, Add, Z_{in}
2 Z_{out}, PC_{in}, Y_{in}, WMFC
3 MDR_{out}, IR_{in}

4 offset X_{out},Select y, Add, Z_{in}
5 Z_{out}, PC_{in}, End.

EX:
The required control steps to fetch and execute <u>a conditional</u> branch(**Br<0**) is illustrated bellow:

Note: N=0 (positive)
N=1 (negative)

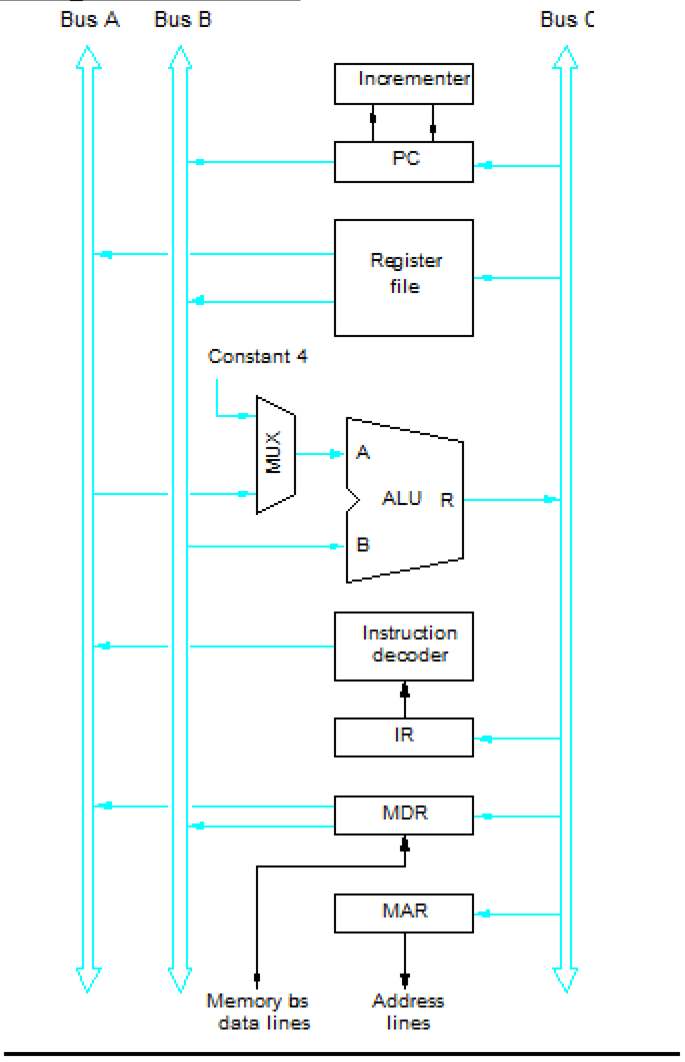
1 PC_{out}, MAR_{in}, Read, Select 4, Add, Z_{in}
2 Z_{out}, PC_{in}, Y_{in}, WMFC
3 MDR_{out}, IR_{in}

4 Branch to 25

25 If N=0 then branch to 1
26 offset X_{out}, Select y, Add, Z_{in}
27 Z_{out}, PC_{in}, End.

H.W= Br > 0

Multiple –Bus Organization:



Example: Write the control steps to fetch and execute the following instruction in multiple- bus CPU:

1 PC$_{out}$, R=B, MAR$_{in}$, Read, Inc PC
2 WMFC
3 MDR$_{out\ \mathbf{B}}$, R=B , IR$_{in}$

4 R4$_{out\ \mathbf{A}}$, R5$_{out\ \mathbf{B}}$, Select A, Add, R6$_{in}$, End.

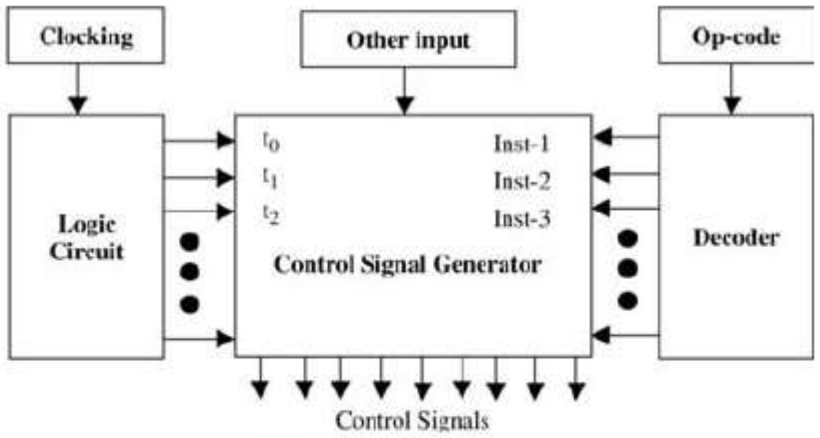
EX: Write the control steps to fetch and execute the following instruction in multiple- bus CPU: **Add (R4), R5,R6**

1 PC_{out}, R=B, MAR_{in}, Read, Inc PC
2 WMFC
3 $MDR_{out\ \mathbf{B}}$, R=B , IR_{in}

4 $R4_{out\ \mathbf{B}}$, R=B, MAR_{in} , Read
5 WMFC
6 $MDR_{out\ \mathbf{B}}$, $R5_{out\ \mathbf{A}}$, Select A, Add, $R6_{in}$, End.

Control Unit:

•The control unit is the main component that directs the system operation by sending control signals to the control buses.
• The system clock produces a continuous sequence of pulses (t0, t1,t2…) in a specified duration and frequency .



Timing of control signals

• The decoder takes the op- code and provide the control signal generator with information about the instruction to be executed.

•The logic circuit module is used with other inputs to generate control signals.

• The signal generator can be specified simply by a set of Boolean equations for its output in terms of its inputs.

There are mainly two different types of control units: **micro programmed** and **hardwired**.

Hardwired control:

• Fixed logic circuits that correspond directly to the Boolean expressions are used to generate the control signals.

• Hardwired control is faster than micro programmed control.

• Hardwired control could be very expensive and complicated for complex systems, but more economical for small systems.

• Hardwired control will require a redesign of the entire systems in the case of any change.

In hardwired control, a direct implementation is accomplished using logic circuits. For each control line, one must find the Boolean expression in terms of the input to the control signal generator .

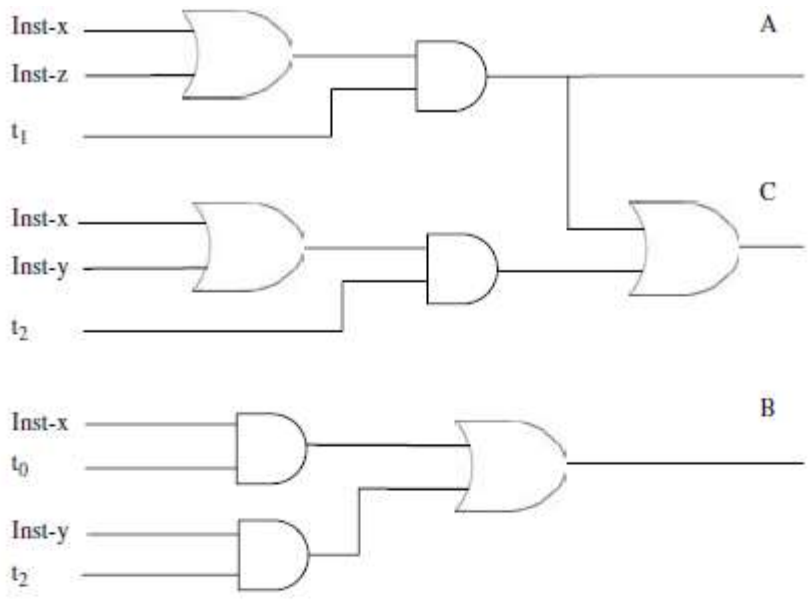
EX: Assume that the instruction set of a machine has the three instructions: Inst-**x**, Inst-**y**, and Inst-**z**; and **A, B, C, D, E, F,G**, and **H** are control lines. The following table shows the control lines that should be activated for the three instructions at the three steps t0 , t1 , and t2 .

Step	Inst-x	Inst-y	Inst-z
t_0	D, B, E	F, H, G	E, H
t_1	C, A, H	G	D, A, C
t_2	G, C	B, C	

The Boolean expressions for control lines A, B, and C can be obtained as follows

$$\begin{aligned} A &= \text{Inst-x} \cdot t_1 + \text{Inst-z} \cdot t_1 = (\text{Inst-x} + \text{Inst-z}) \cdot t_1 \\ B &= \text{Inst-x} \cdot t_0 + \text{Inst-y} \cdot t_2 \\ C &= \text{Inst-x} \cdot t_1 + \text{Inst-x} \cdot t_2 + \text{Inst-y} \cdot t_2 + \text{Inst-z} \cdot t_1 \\ &= (\text{Inst-x} + \text{Inst-z}) \cdot t_1 + (\text{Inst-x} + \text{Inst-y}) \cdot t_2 \end{aligned}$$

The figure below shows the logic circuits for these control lines. Boolean expressions for the rest of the control lines can be obtained in a similar way.



Logic circuits for control lines A, B, and C

Micro programmed Control Unit

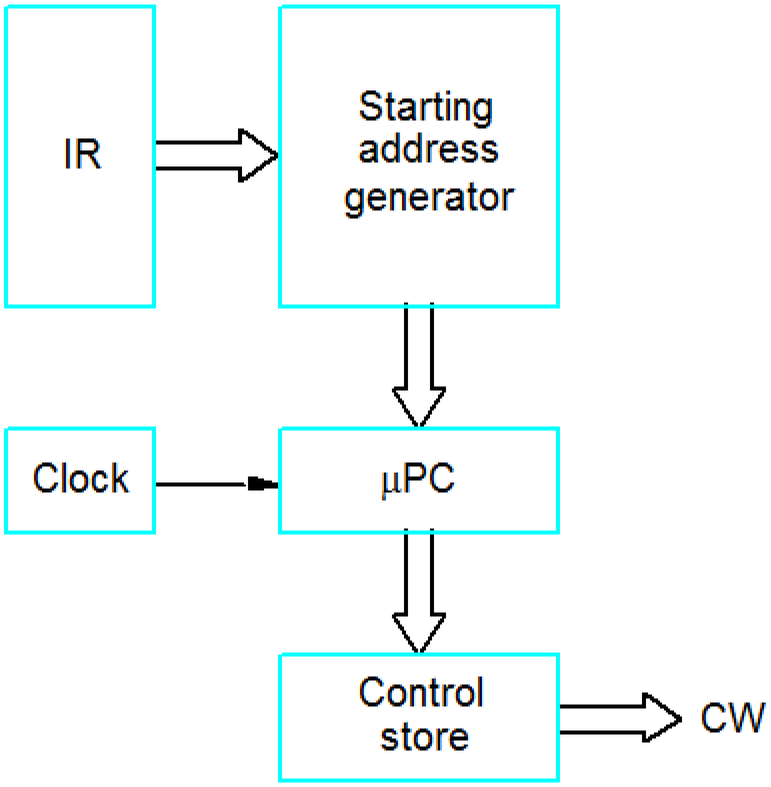
• The idea of micro programmed control is to store the control signals associated with the implementation of an instruction as a **micro program** in a special memory called a **control memory (CM)**.

• A **control word** is a microinstruction that specifies one or more micro operations(control signals).

• A sequence of microinstructions is called a ***micro program***

• It should also be noted that micro programmed control could adapt easily to changes in the system design. We can easily add new instructions without changing hardware

• A **microinstruction** is a vector of bits, where each bit is a control signal, condition code, or the address of the next microinstruction.



• When an instruction is fetched from memory, the op-code is mapped to a microinstruction address in the control memory.

• The microinstruction processor uses that address to fetch the first microinstruction in the micro program.

•After fetching each microinstruction, the appropriate control lines will be enabled. Every control line that corresponds to a "1" bit should be turned on. Every control line that corresponds to a "0" bit should be left off.

•After completing the execution of one microinstruction, a new microinstruction will be fetched and executed .In the following is an example of a **micro program**:

1 PC$_{out}$, MAR$_{in}$, Read, Select 4, Add, Z$_{in}$
2 Z$_{out}$, PC$_{in}$, Y$_{in}$, WMFC
3 MDR$_{out}$, IR$_{in}$

4 Branch to 25

25 If N=0 then branch to 1
26 offset X$_{out}$, Select y, Add, Z$_{in}$
27 Z$_{out}$, PC$_{in}$, End.

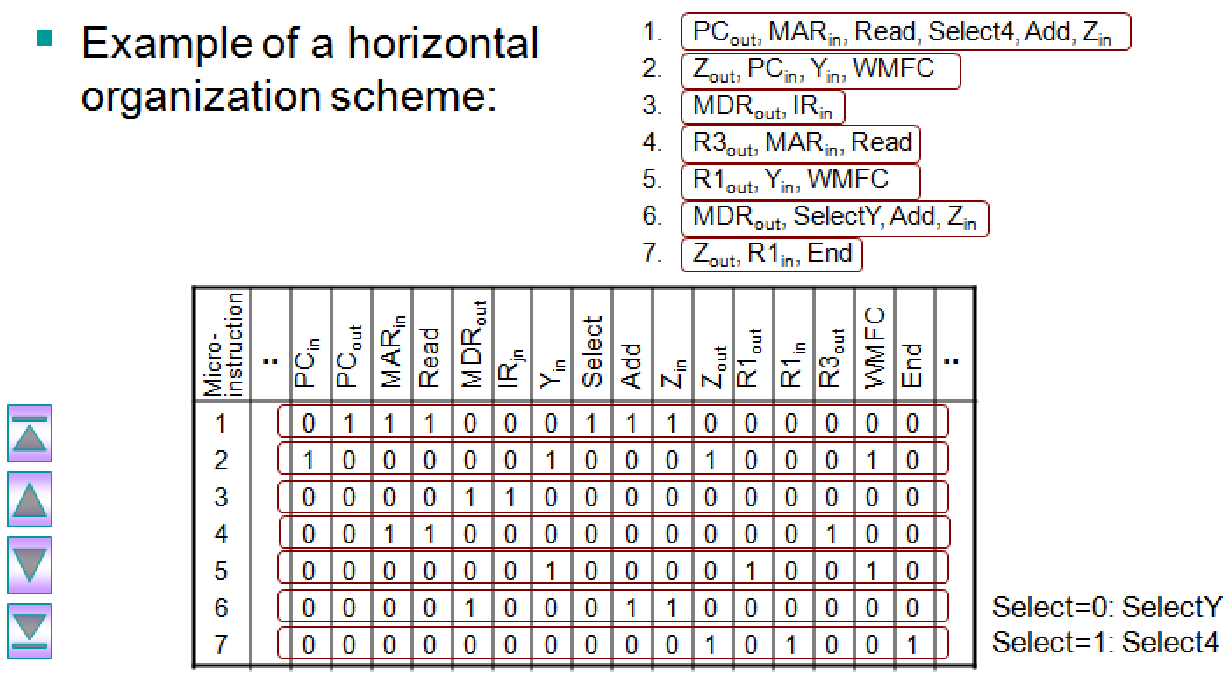
Horizontal Versus Vertical Microinstructions

Micro instructions can be classified as *horizontal* or *vertical*:

Individual bits in **horizontal** microinstructions correspond to individual control lines. Horizontal microinstructions are long and allow maximum parallelism since each bit controls a single control line.

In **vertical** microinstructions, control lines are coded into specific fields within a microinstruction. Decoders are needed to map a field of k bits to 2^k possible combinations of control lines. For example, a 3-bit field in a microinstruction could be used to specify any one of eight possible lines.

Because of the encoding, vertical microinstructions are much shorter than horizontal ones. Control lines encoded in the same field cannot be activated simultaneously. Therefore, vertical microinstructions allow only limited parallelism.



- Example of a horizontal organization scheme:

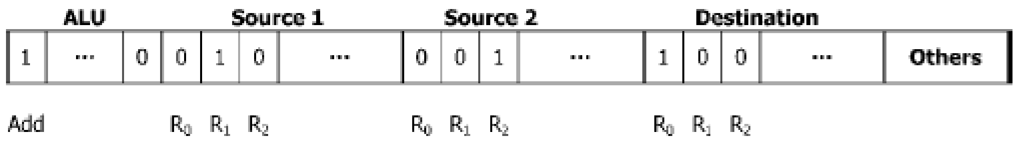
1. PC_{out}, MAR_{in}, Read, Select4, Add, Z_{in}
2. Z_{out}, PC_{in}, Y_{in}, WMFC
3. MDR_{out}, IR_{in}
4. $R3_{out}$, MAR_{in}, Read
5. $R1_{out}$, Y_{in}, WMFC
6. MDR_{out}, SelectY, Add, Z_{in}
7. Z_{out}, $R1_{in}$, End

Micro-instruction	..	PC_{in}	PC_{out}	MAR_{in}	Read	MDR_{out}	IR_{in}	Y_{in}	Select	Add	Z_{in}	Z_{out}	$R1_{out}$	$R1_{in}$	$R3_{out}$	WMFC	End	..
1		0	1	1	1	0	0	0	1	1	1	0	0	0	0	0	0	
2		1	0	0	0	0	0	1	0	0	0	1	0	0	0	1	0	
3		0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	
4		0	0	1	1	0	0	0	0	0	0	0	0	0	1	0	0	
5		0	0	0	0	0	0	1	0	0	0	0	1	0	0	1	0	
6		0	0	0	0	1	0	0	0	1	1	0	0	0	0	0	0	
7		0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	1	

Select=0: SelectY
Select=1: Select4

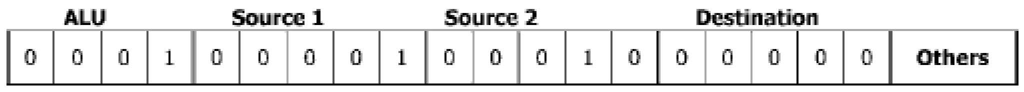
Example: Consider the three-bus data path shown in the figure below. In addition to the PC, IR, MAR, and MDR, assume that there are **16** general-purpose registers numbered R0–R15 . Also, assume that the ALU supports **eight** functions (add, subtract, multiply, divide, AND, OR, shift left, and shift right). Consider the add operation **Add R1 , R2 , R0** , which adds the contents of source registers R1 , R2 , and store the results in destination register R0 .

The format of the microinstruction under **horizontal** organization:



Microinstruction for Add R_1, R_2, R_0

The format of the microinstruction under **vertical** organization:



Microinstruction for Add R_1, R_2, R_0