Computer Architecture and Operating System

Prof. Indranil Sengupta

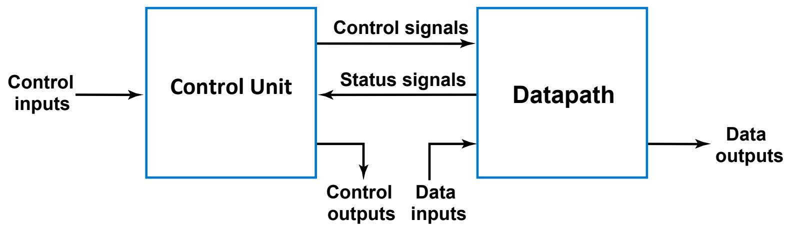
Department of Computer Science and Engineering

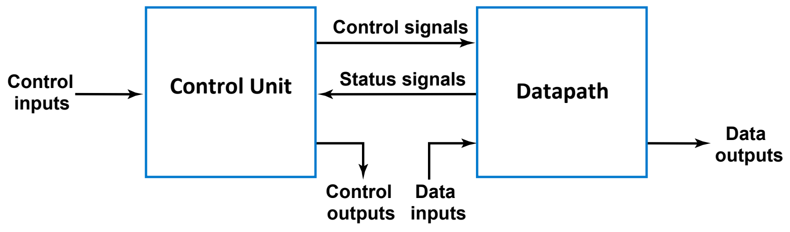
IIT Kharagpur

Design of Control Unit

Basic Concept

- Any digital circuit module consists of two components:
  - **a) Data Path** – performs data transfer and processing operations.
    (Adder, multiplier, multiplexer, register, bus, counter, decoder, etc.)
  - **b) Control Unit** – generates control signals for the data path for sequencing of the operations.
    (Basically a finite-state machine)





- The control unit receives:
  - External control inputs
  - Status signals generated by the datapath
- The control unit sends:
  - Control signals for activating the datapath components
  - Control outputs

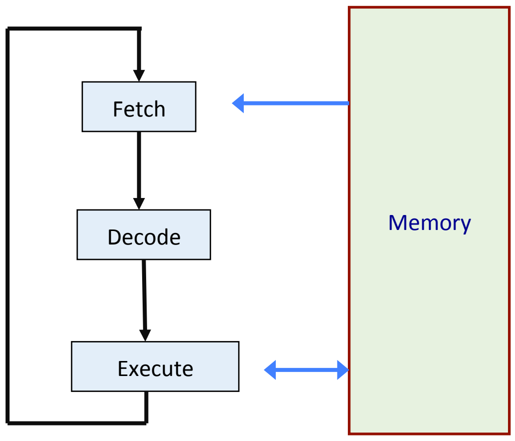
Types of Control Units

- Two distinct classes:
  a) Non-programmable or ***hardwired***
  b) Programmable or ***microprogrammed***
- A hardwired control unit has dedicated circuits, and does not fetch or sequence instructions from any memory.
- A microprogrammed control unit has a *separate memory* where the information regarding control signals and sequencing is stored.
  - For example, a *micro-program counter* (μPC) points to the next set of control signals (microinstruction word) to be fetched from memory.

Recall how an instruction gets executed

```
repeat forever
      // till power off or
      // system failure
{
    Fetch instruction
    Decode instruction
    Execute instruction
}
```

Fetch-Decode-Execute Cycle



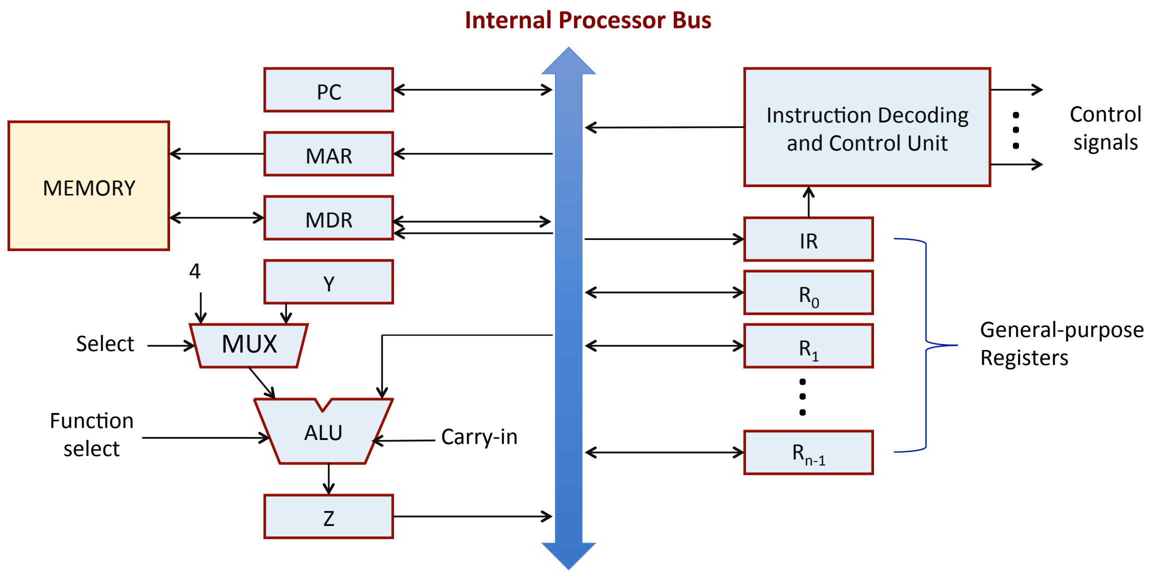
Requirements for Executing an Instruction

- Example: `ADD R1,R2,R3`
- Requirements for understanding the mechanism:
  - The necessary registers must be present.
  - The internal organization of the registers must be known.
  - The data path must be known.
  - For instruction execution, a number of *micro-operations* are carried out on the data path.
    - An instruction consists of several *micro-operations* or *micro-instructions*.
    - May involve movement of data.

Kinds of Data Movement Inside the CPU

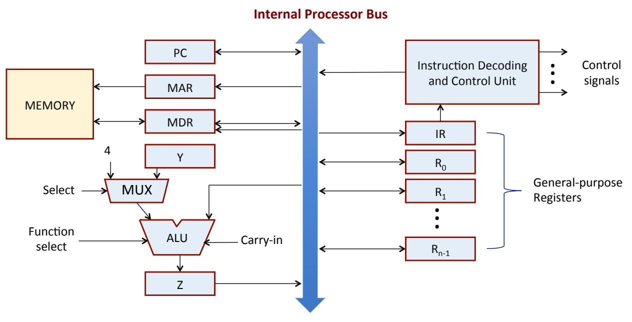
- Broadly three types:
  a) Register to Register
  b) Register to ALU
  c) ALU to Register
- Data movement must be supported in the data path by:
  - The Registers
  - The Bus (single or multiple)
  - Some temporary registers, multiplexers, etc.

Single Bus Organization

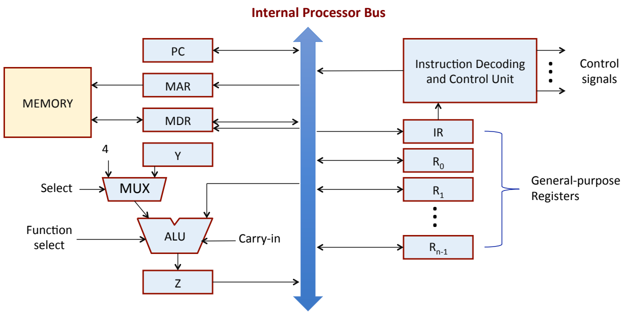


Example: Single Internal Bus Organization

- All the registers and various units are connected using a single internal bus.
- Registers R_0-R_{n-1} are *general-purpose registers* used for various purposes.
- Y and Z are used for storing intermediate results and never used by instructions explicitly.
- The multiplexer selects either a constant 4 or output of register Y.
  - When PC is incremented, a constant 4 has to be added.

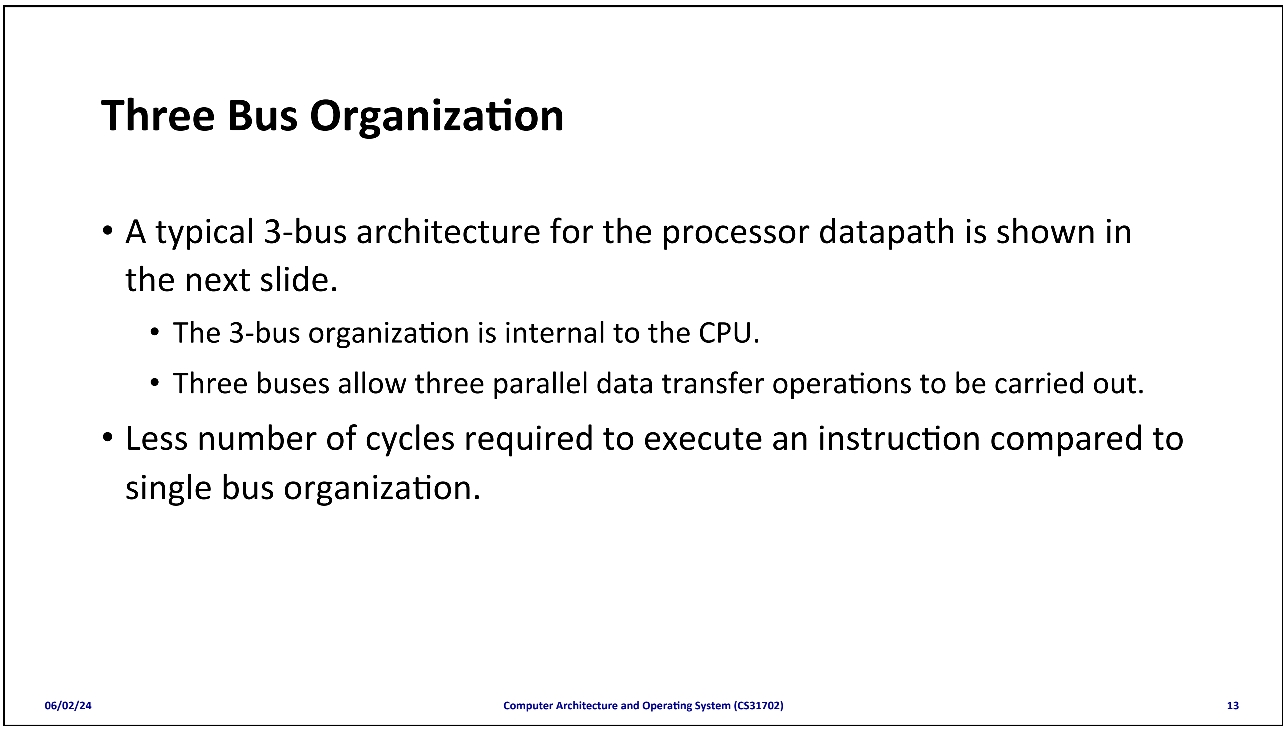


- The instruction decoder and control unit is responsible for performing the actions specified by the instruction loaded into IR.
- The decoder generates all the control signals in the proper sequence required to execute the instruction specified by the IR.
- The registers, the ALU and the interconnecting bus are collectively referred to as the *datapath*.



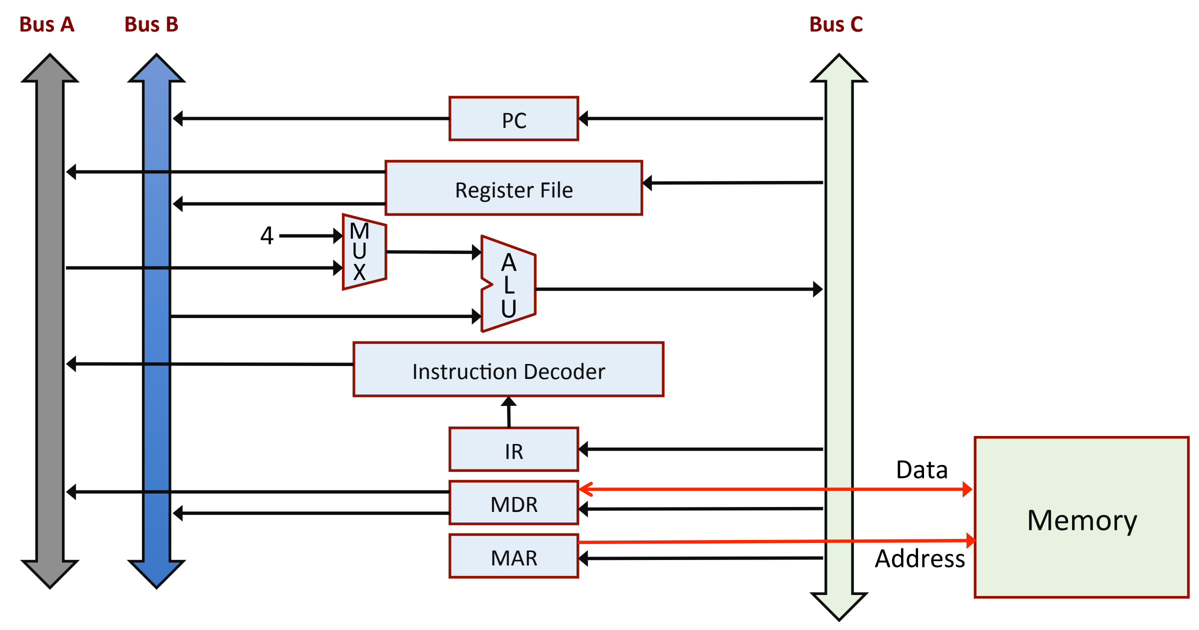
Kinds of Operations

- Transfer of data from one register to another.
  MOVE R1, R2 // R1 = R2
- Perform arithmetic or logic operation on data loaded into registers.
  ADD R1, R2 // R1 = R1 + R2
- Fetch the content of a memory location and load it into a register.
  LOAD R1, LOCA // R1 = Mem[LOCA]
- Store a word of data from a register into a given memory location.
  STORE LOCA, R1 // Mem[LOCA] = R1



Three Bus Organization

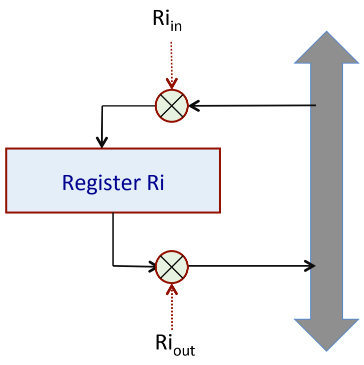
- A typical 3-bus architecture for the processor datapath is shown in the next slide.
  - The 3-bus organization is internal to the CPU.
  - Three buses allow three parallel data transfer operations to be carried out.
- Less number of cycles required to execute an instruction compared to single bus organization.

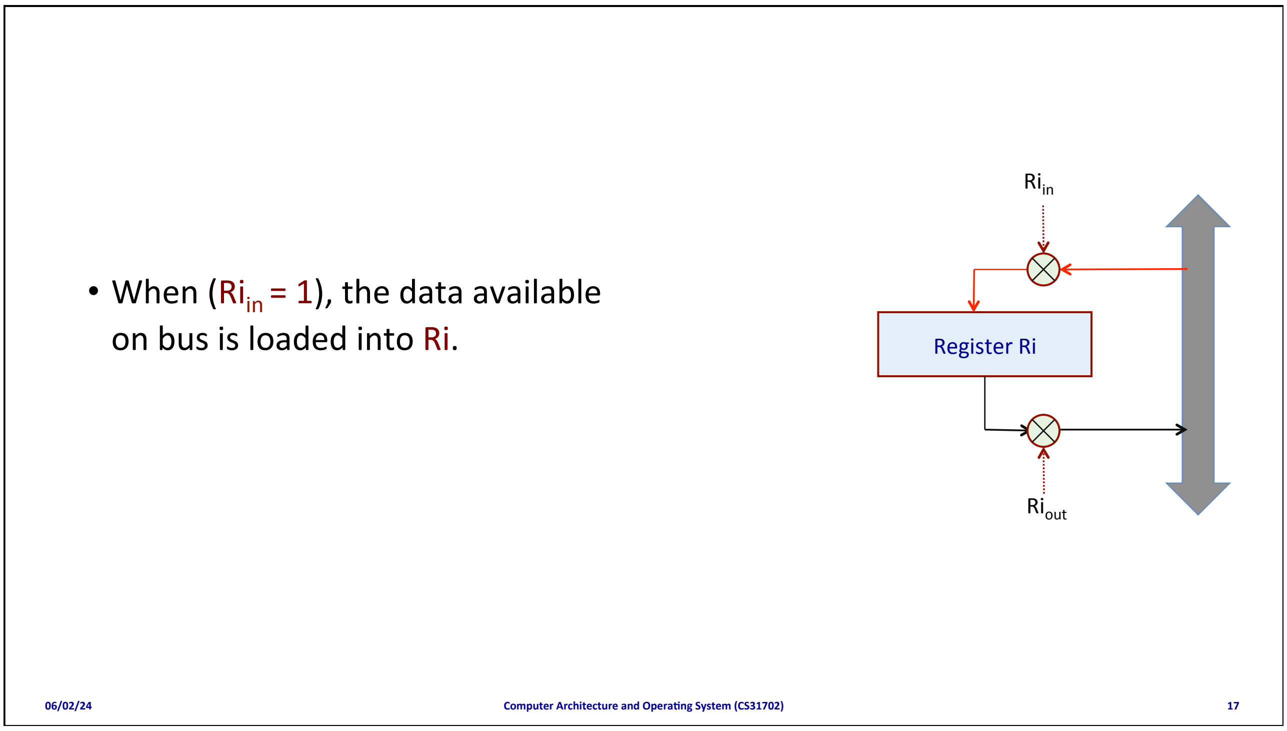


Control Signals for Register-Register and Register-Memory Transfers

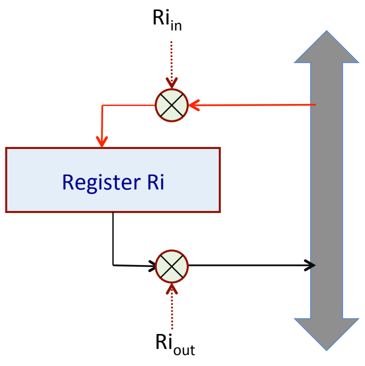
Organization of a Register

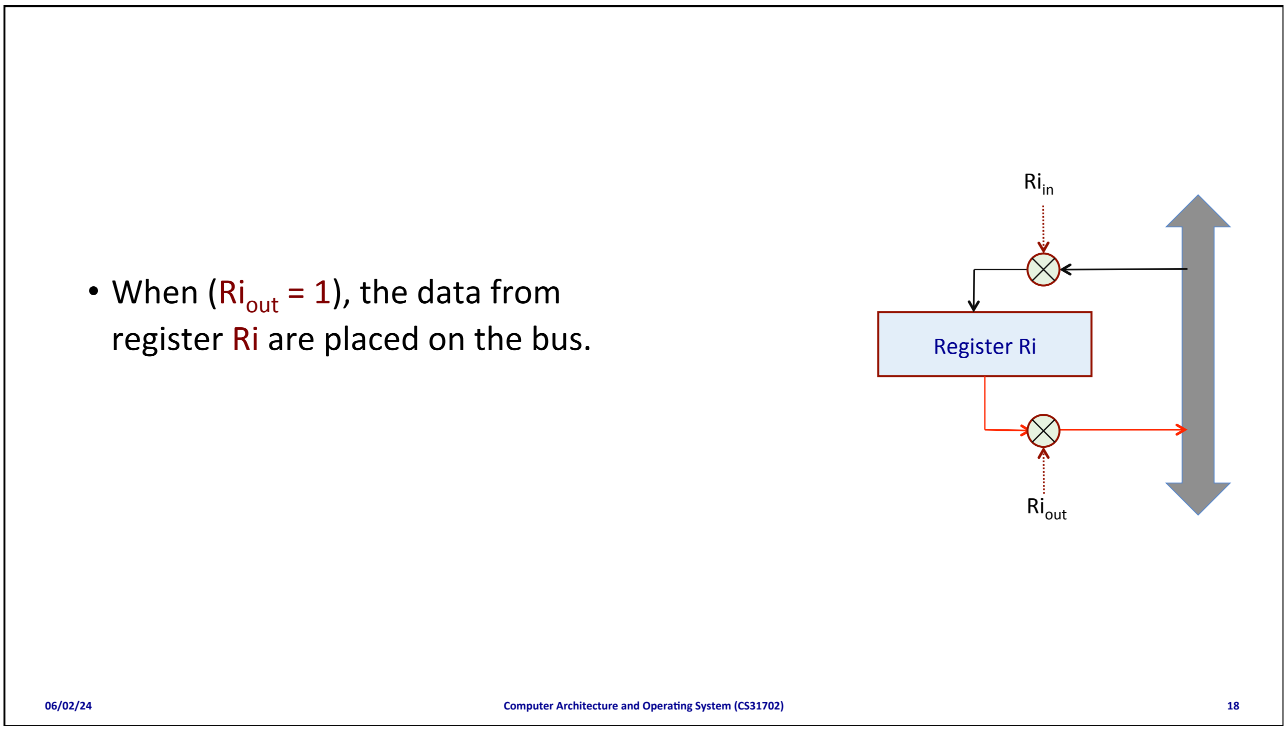
- A register is used for temporary storage of data (parallel-in, parallel-out, etc.).
- A register Ri typically has two control signals.
  - Ri_{in} : used to load the register with data from the bus.
  - Ri_{out} : used to place the data stored in the register on the bus.
- Input and output lines of the register Ri are connected to the bus via controlled switches.



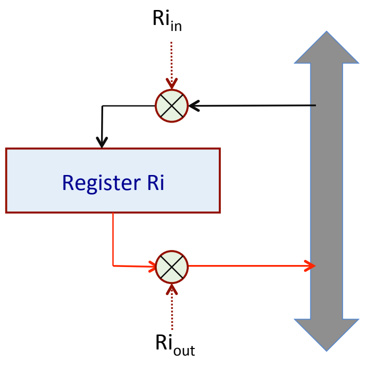


- When (Ri_{in} = 1), the data available on bus is loaded into Ri.





- When (Ri_{out} = 1), the data from register Ri are placed on the bus.



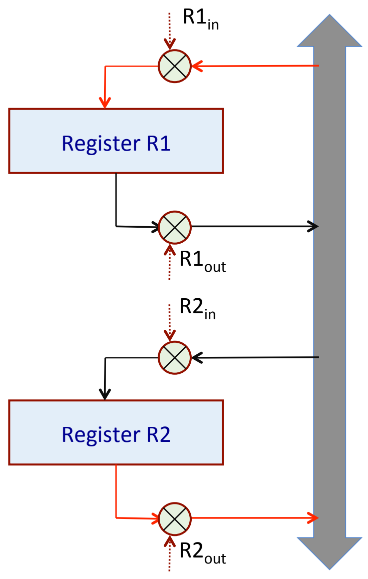
Register-Register Transfer

MOVE R1, R2 // R1 = R2

- Enable the output of R2 by setting $R2_{out} = 1$.
- Enable the input of register R1 by setting $R1_{in} = 1$.
- All operations are performed in synchronism with the processor clock.
  - The control signals are asserted at the start of the clock cycle.
  - After data transfer the control signals will return to 0.
- We write as *T1: $R2_{out}$, $R1_{in}$*

Time Step

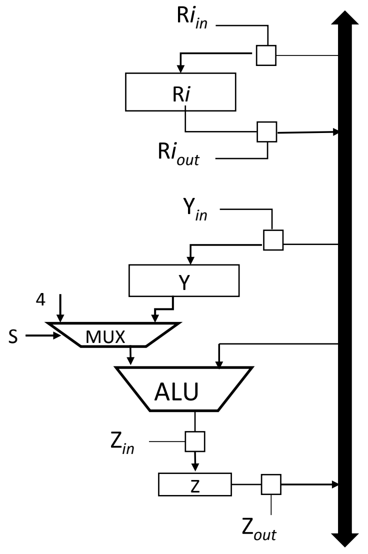
Control Signals



ALU Operation

ADD R1, R2 // R1 = R1 + R2

- Bring the two operands (R1 and R2) to the two inputs of the ALU.
  One through Y (R1) and another (R2) directly from internal bus.
- Result is stored in Z and finally transferred to R1.
  T1: $R1_{out}$, Y_{in}
  T2: $R2_{out}$, SelectY, ADD, Z_{in}
  T3: Z_{out}, $R1_{in}$



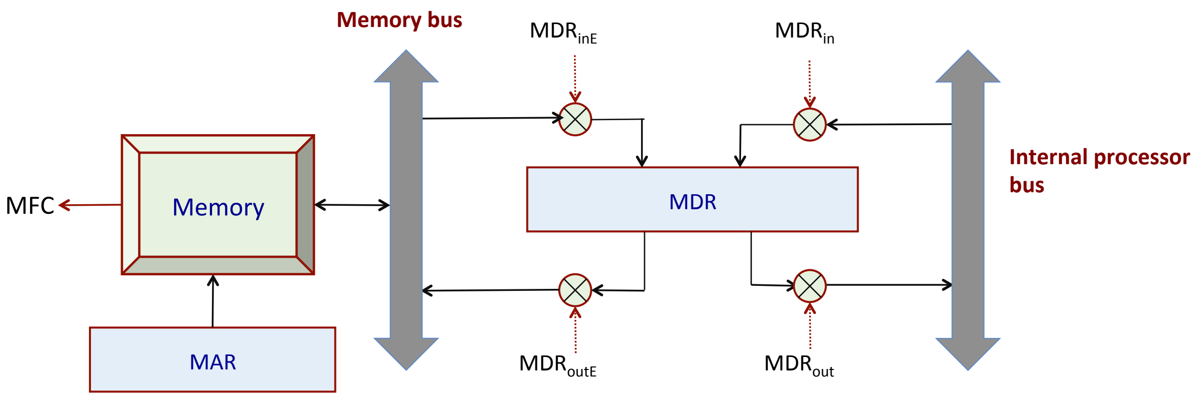
Fetching a Word from Memory

- The steps involved to fetch a word from memory:
  - The processor specifies the address of the memory location in *MAR* where the data or instruction is stored.
  - The processor requests a *Read* operation.
  - The information to be fetched can either be an instruction or an operand of the instruction.
  - The data read is brought from the memory to *MDR*.
  - Then it can be transferred to the required register or ALU for further operation.

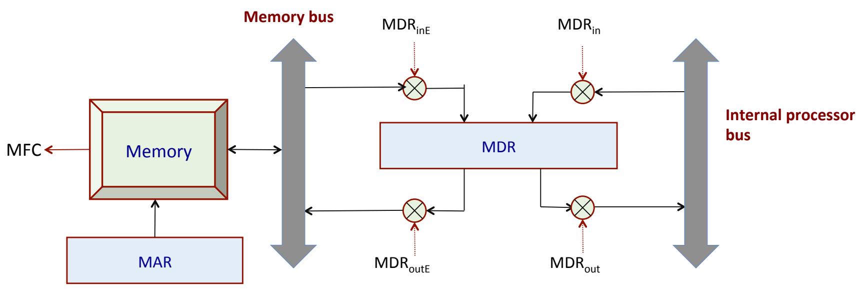
Storing a Word into Memory

- The steps involved to store a word into the memory:
  - The processor specifies the address of the memory location in *MAR* where the data is to be written.
  - The data to be written in loaded into *MDR*.
  - The processor requests a *Write* operation.
  - The content of *MDR* will be written to the specified memory location.

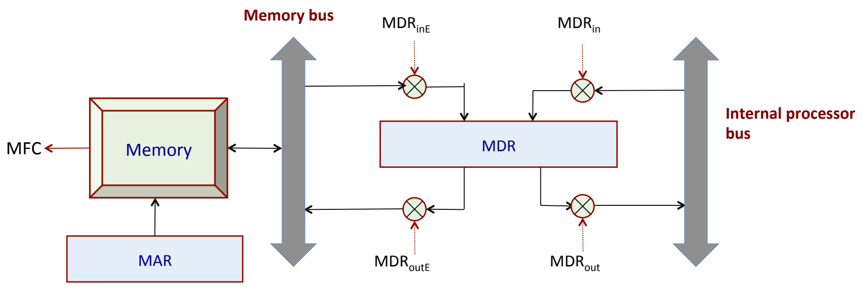
Connecting MDR to Memory Bus and Internal Bus



- **Memory read/write operation:**
  - The address of memory location is transferred to *MAR*.
  - At the same time a read/write control signal is provided to indicate the operation.
  - For read the data from memory data bus comes to *MDR* by activating MDR$_{inE}$.
  - For write the data from *MDR* goes to memory data bus by activating the signal MDR$_{outE}$.



- When the processor sends a read request, it has to wait until the data is read from the memory and written into *MDR*.
- To accommodate the variability in response time, the processor has to wait until it receives an indication from the memory that the read operation has been completed.
- A control signal called *Memory Function Complete* (MFC) is used for this purpose.
  - When this signal is 1, indicates that the content of the specified location is read and are available on the data line of the memory bus.
  - Then the data can be made available to *MDR*.



Fetch a word: MOVE R1, (R2)

1. MAR ← R2
2. Start a Read operation on the memory bus
3. Wait for the MFC response from the memory
4. Load MDR from the memory
5. R1 ← MDR

Control steps:

a) R2$_{out}$, MAR$_{in}$, Read
b) MDR$_{inE}$, WMFC
c) MDR$_{out}$, R1$_{in}$

Store a word: MOVE (R1), R2

1. MAR ← R1
2. MDR ← R2
3. Start a Write operation on the memory bus
4. Wait for the MFC response from the memory

Control steps:

a) $R1_{out}$, MAR_{in}

b) $R2_{out}$, MDR_{in}, Write

c) MDR_{outE}, WMFC

Execution of a Complete Instruction

ADD R1, R2 // R1 = R1 + R2

T1: PC_{out}, MAR_{in}, Read, Select4, ADD, Z_{in}

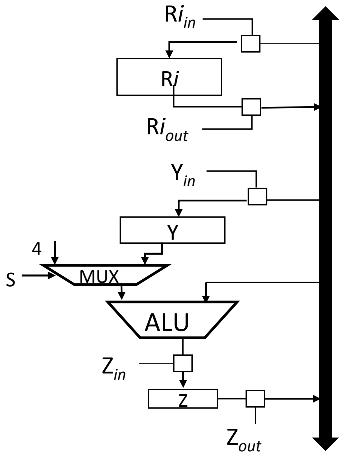
T2: Z_{out}, PC_{in}, Y_{in}, WMFC

T3: MDR_{out}, IR_{in}

T4: $R1_{out}$, Y_{in}, SelectY

T5: $R2_{out}$, ADD, Z_{in}

T6: Z_{out}, $R1_{in}$



Example for a Three Bus Organization

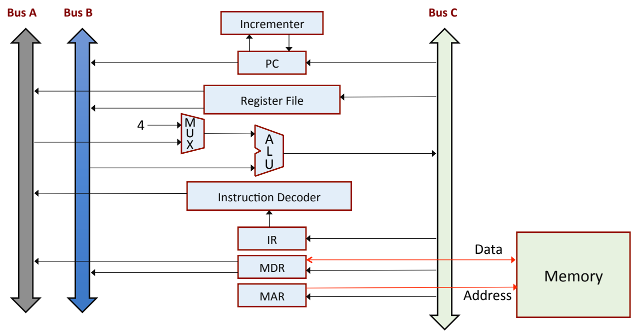
SUB R1, R2, R3 // R1 = R2 – R3

T1: PC_{out}, R = B, MAR_{in} , READ, IncPC

T2: WMFC

T3: MDR_{outB} , R = B, IR_{in}

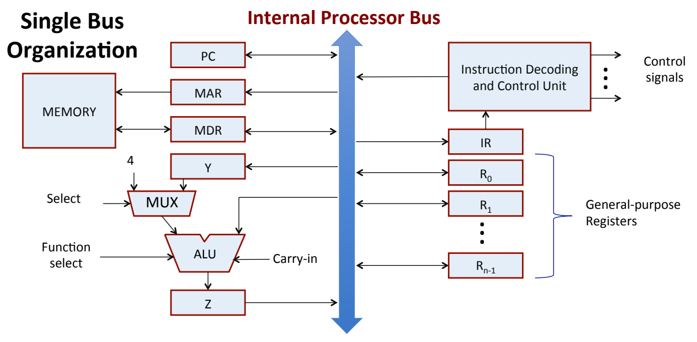
T4: $R2_{outA}$, $R3_{outB}$, SelectA, SUB, $R1_{in}$, End



Example ISA for a Single-Bus Processor Architecture: Micro-operations

Introduction

- We select a set of 12 instructions.
- Discuss the control signals required to execute these instructions on the single-bus processor architecture.
- Assumptions:
  - Memory is byte oriented.
  - Instruction size is 32 bits (4 bytes).

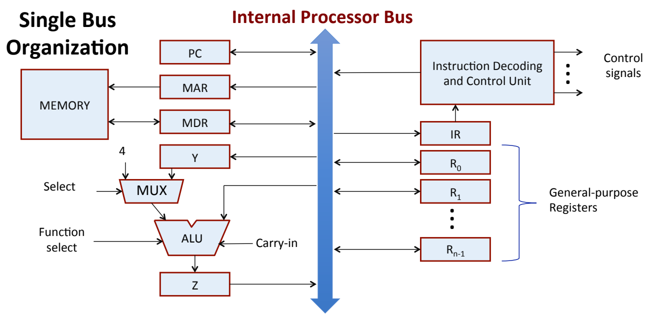


Various instructions: Control sequence

1.	ADD R1, R2	// R1 = R1 + R2
2.	ADD R1, LOCA	// R1 = R1 + Mem[LOCA]
3.	LOAD R1, LOCA	// R1 = Mem[LOCA]
4.	STORE LOCA, R1	// Mem[LOCA] = R1
5.	MOVE R1, R2	// R1 = R2
6.	MOVE R1, #10	// R1 = 10
7.	BR LOCA	// PC = LOCA
8.	BZ LOCA	// PC = LOCA if Zero flag is set
9.	INC R1	// R1 = R1 + 4
10.	DEC R1	// R1 = R1 – 4
11.	CMP R1, R2	// R1 – R2; set the flags
12.	HALT	// Machine Halt

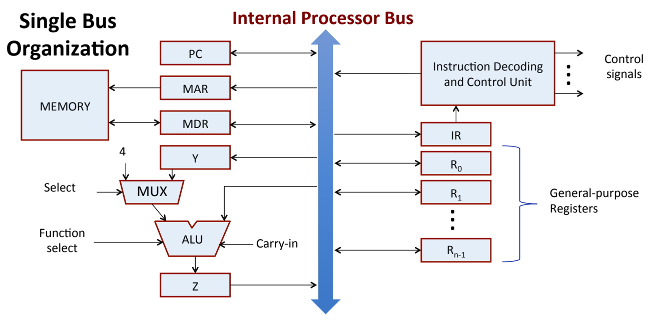
1. ADD R1, R2 (R1 = R1 + R2)

Steps	Action
1	PC_{out}, MAR_{in}, Read, Select4, Add, Z_{in}
2	Z_{out}, PC_{in}, WMFC
3	MDR_{out}, IR_{in}
4	$R1_{out}$, Y_{in}
5	$R2_{out}$, SelectY, Add, Z_{in}
6	Z_{out}, $R1_{in}$, End



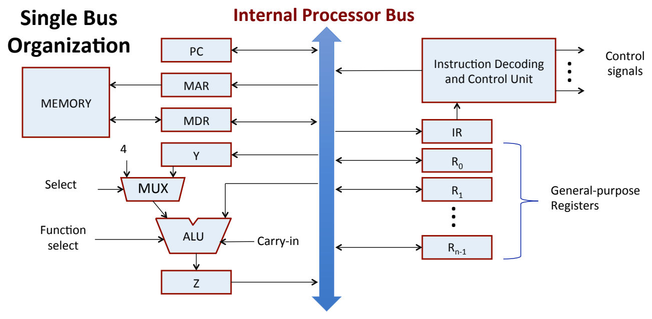
2. ADD R1, LOCA (R1 = R1 + Mem[LOCA])

Steps	Action
1	PC_{out}, MAR_{in}, Read, Select4, Add, Z_{in}
2	Z_{out}, PC_{in}, WMFC
3	MDR_{out}, IR_{in}
4	Address field of IRout, MAR_{in}, Read
5	$R1_{out}$, Y_{in}, WMFC
6	MDR_{out}, SelectY, Add, Z_{in}
7	Z_{out}, $R1_{in}$, End



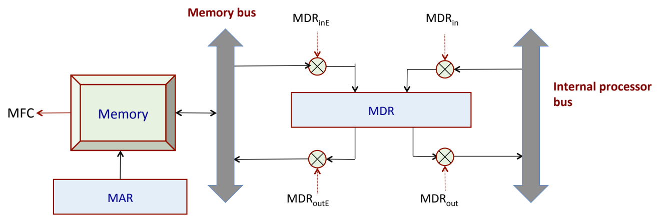
3. LOAD R1, LOCA (R1 = Mem[LOCA])

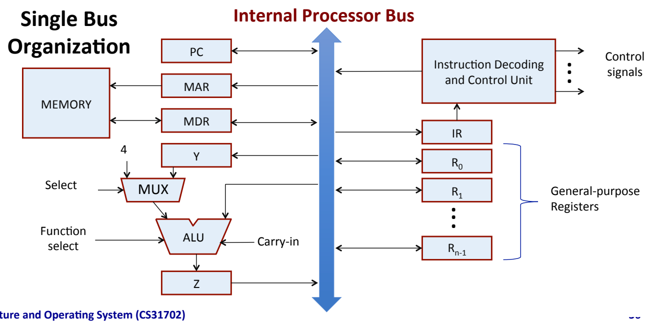
Steps	Action
1	PC_{out}, MAR_{in}, Read, Select4, Add, Z_{in}
2	Z_{out}, PC_{in}, WMFC
3	MDR_{out}, IR_{in}
4	Address field of IRout, MAR_{in}, Read
5	WMFC
6	MDR_{out}, $R1_{in}$, END



4. STORE LOCA, R1 (Mem[LOCA] = R1)

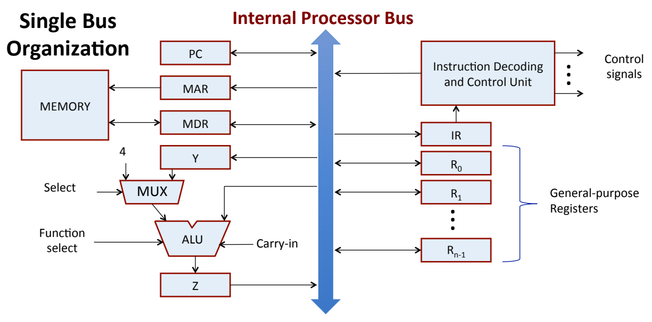
Steps	Action
1	PC_{out}, MAR_{in}, Read, Select4, Add, Z_{in}
2	Z_{out}, PC_{in}, WMFC
3	MDR_{out}, IR_{in}
4	Address field of IRout, MAR_{in}
5	$R1_{out}$, MDR_{in}, Write
6	MDR_{outE}, WMFC, End





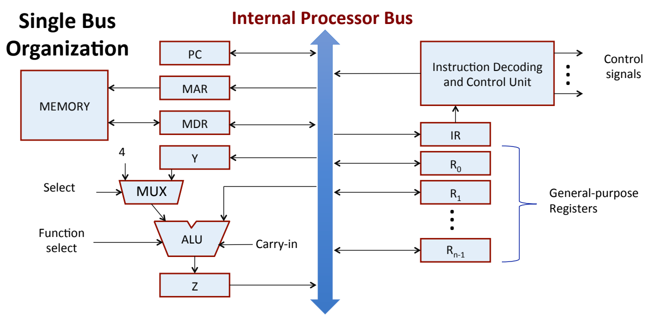
5. MOVE R1, R2 (R1 = R2)

Steps	Action
1	PC_{out}, MAR_{in}, Read, Select4, Add, Z_{in}
2	Z_{out}, PC_{in}, WMFC
3	MDR_{out}, IR_{in}
4	$R2_{out}$, $R1_{in}$, END



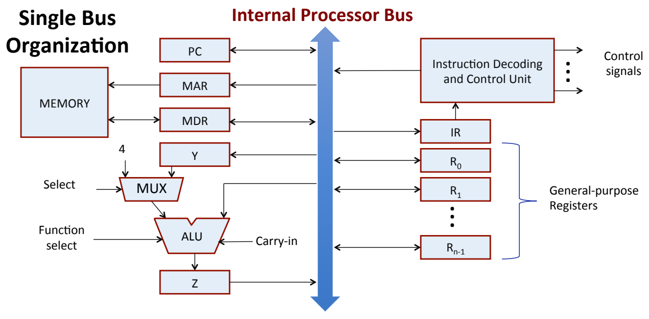
6. MOVE R1, #10 (R1 = 10)

Step	Action
1	PC_{out}, MAR_{in}, Read, Select4, Add, Z_{in}
2	Z_{out}, PC_{in}, WMFC
3	MDR_{out}, IR_{in}
4	Immediate field of IR_{out}, $R1_{in}$, END



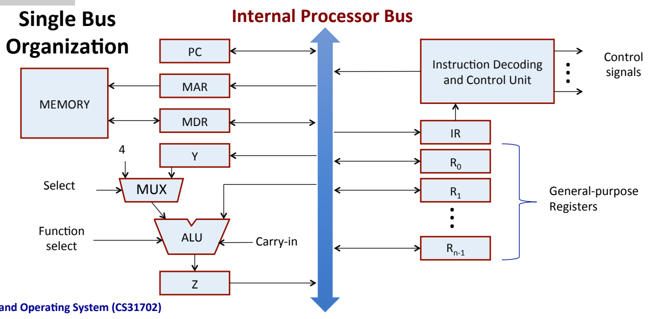
7. BRANCH Label (PC = PC + offset)

Step	Action
1	PC_{out}, MAR_{in}, Read, Select4, Add, Z_{in}
2	Z_{out}, PC_{in}, Y_{in}, WMFC
3	MDR_{out}, IR_{in}
4	Offset-field-of-IR_{out}, SelectY, Add, Z_{in}
5	Z_{out}, PC_{in}, End



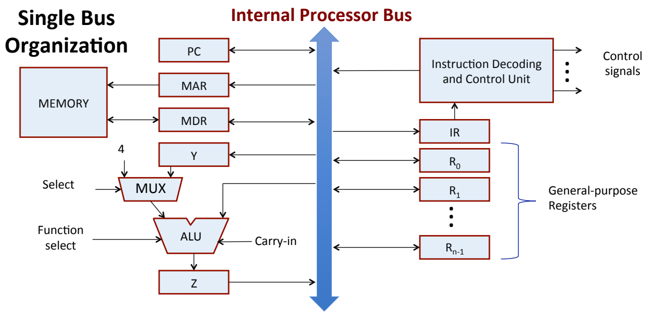
8. BZ Label (if Z=1 PC = PC + offset)

Step	Action
1	PC_{out}, MAR_{in}, Read, Select4, Add, Z_{in}
2	Z_{out}, PC_{in}, Y_{in}, WMFC
3	MDR_{out}, IR_{in}
4	Offset-field-of-IR_{out}, SelectY, Add, Z_{in}, If Z=0 then End
5	Z_{out}, PC_{in}, End



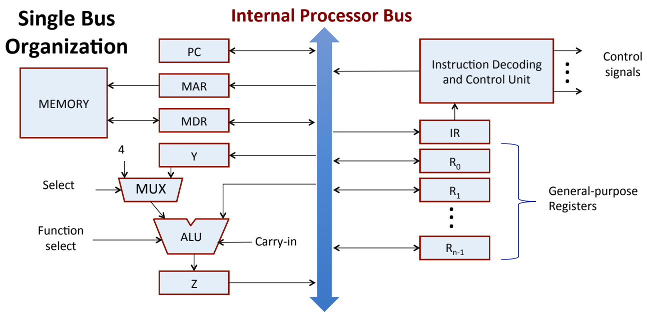
9. INC R1 (R1 = R1 + 4)

Steps	Action
1	PC_{out}, MAR_{in}, Read, Select4, Add, Z_{in}
2	Z_{out}, PC_{in}, WMFC
3	MDR_{out}, IR_{in}
4	$R1_{out}$, Select4, Add, Z_{in}
5	Z_{out}, $R1_{in}$, End



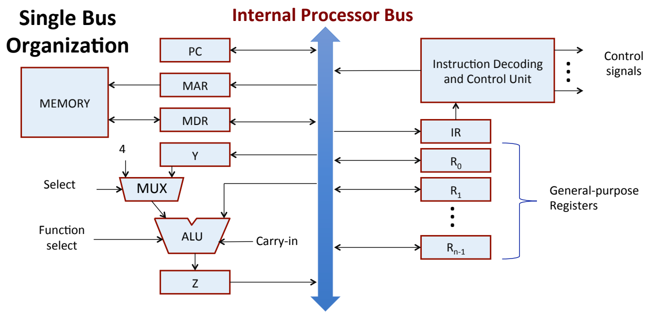
10. DEC R1 (R1 = R1 – 4)

Steps	Action
1	PC_{out}, MAR_{in}, Read, Select4, Add, Z_{in}
2	Z_{out}, PC_{in}, WMFC
3	MDR_{out}, IR_{in}
4	$R1_{out}$, Select4, SUB, Z_{in}
5	Z_{out}, $R1_{in}$, End



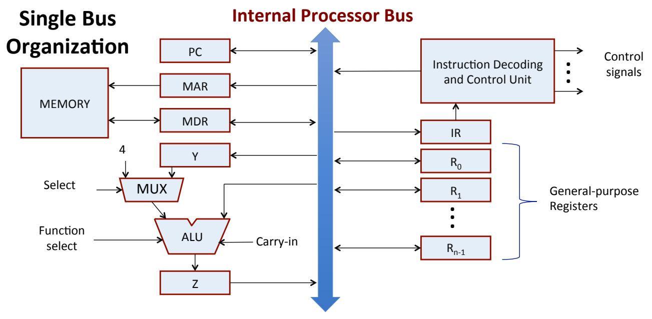
11. CMP R1, R2

Steps	Action
1	PC_{out}, MAR_{in}, Read, Select4, Add, Z_{in}
2	Z_{out}, PC_{in}, WMFC
3	MDR_{out}, IR_{in}
4	$R1_{out}$, Y_{in}
5	$R2_{out}$, SelectY, Sub, Z_{in}, End



12. HALT

Steps	Action
1	PC_{out}, MAR_{in}, Read, Select4, Add, Z_{in}
2	Z_{out}, PC_{in}, WMFC
3	MDR_{out}, IR_{in}
4	End



Hardwired and Microprogrammed Control Unit Design

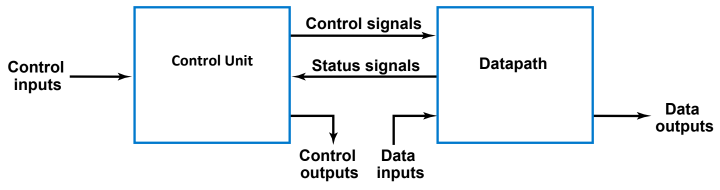
Introduction

- To execute an instruction, the processor must generate control signals for the data path in proper sequence.
  - Example: ADD R1, R2

    T1: $R1_{out}$, Y_{in}, SelectY

    T2: $R2_{out}$, ADD, Z_{in}

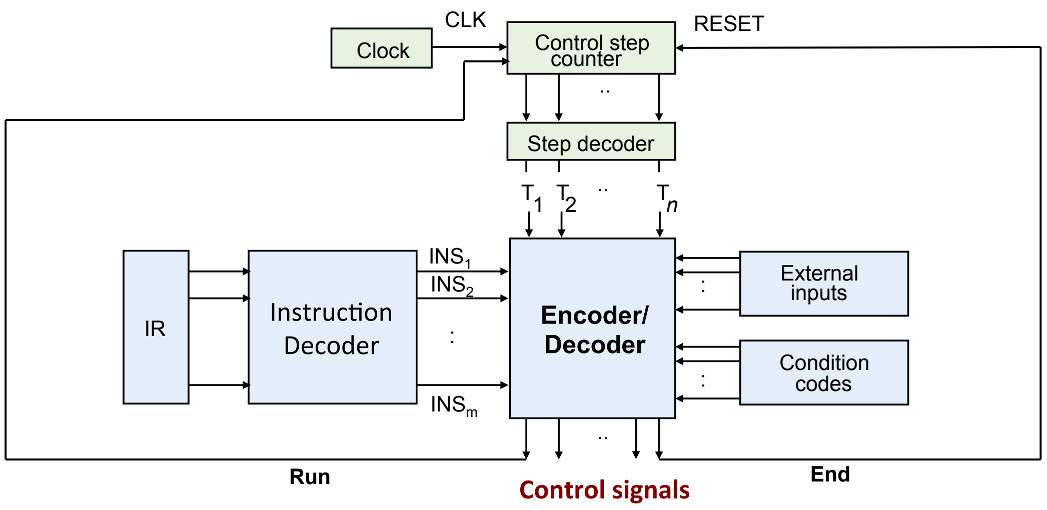
    T3: Z_{out}, $R1_{in}$



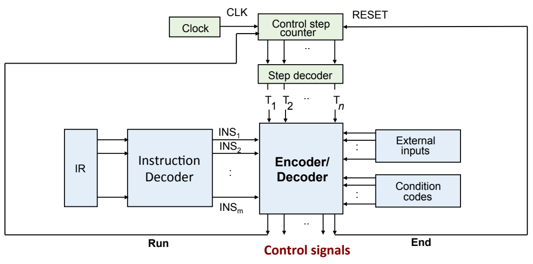
- Two alternate approaches:
  1. Hardwired control unit design
  2. Microprogrammed control unit design

(a) Hardwired Control Unit

- Use a specialized hardware to generate the control signals corresponding the the different instructions.
  - Optimized and fast.
  - Lacks flexibility → difficult to make changes.
- We discuss one approach for designing such a control unit.
  - Based on a step counter / decoder to identify the time steps.

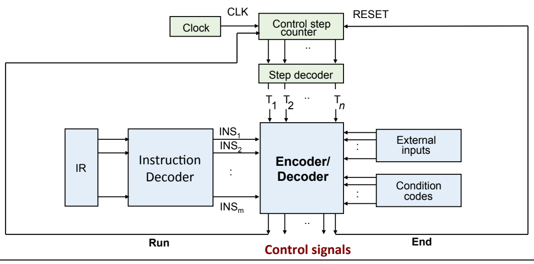


The Basic Idea

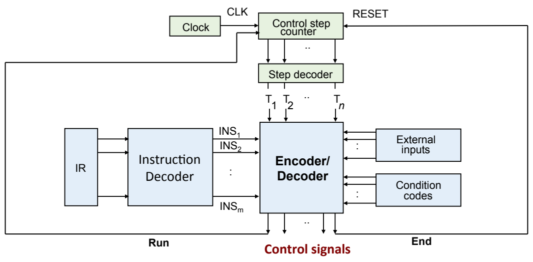


- Assumption:
  - Each step in the sequence can be completed in one clock cycle.
- A counter is used to keep track of the time step (T_1, T_2, T_3, etc.).
- The control signals are determined by the following information:
  - Content of control step counter (i.e., which time step)
  - Content of instruction register (i.e., which instruction)
  - Content of conditional code flags
  - External input signals such as MFC (Memory Function Complete)

- The encoder/decoder is a combinational circuit which generates control signals depending on the inputs provided.
- The step decoder generates separate signal line for each step in the control sequence (T_1, T_2, T_3, etc.).
  - Depending on maximum steps required for an instruction, the step decoder is designed.
  - If a maximum of 10 steps are required, then a 4 x 16 step decoder is used.
- Among the total set of instructions, the instruction decoder is used to select one of them. (That particular line will be 1 and rest will be 0).
  - If a maximum of 100 instructions are present in the ISA then a 7 x 128 instruction decoder is used.



- At every clock cycle the RUN signal is used to increment the counter by one.
  - When RUN is 0 the counter stops counting.
  - This signal is needed when WMFC is issued.
- END signal starts a new instruction.
  - It resets the control step counter to its starting value.
- The sequence of operations carried out by the control unit is determined by the wiring of the logic elements and hence it is named *hardwired*.
- This approach of control unit design is fast but limited to the complexity of instruction set that is implemented.



Generation of Control Signals (Example with 3 instructions)

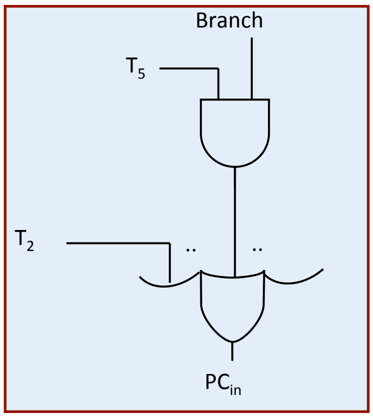
	ADD R1, R2
1	PC_{out}, MAR_{in}, Read, Select4, Add, Z_{in}
2	Z_{out}, Pc_{in}, WMFC
3	MDR_{out}, IR_{in}
4	$R1_{out}$, Y_{in}
5	$R2_{out}$, SelectY, Add, Z_{in}
6	Z_{out}, $R1_{in}$, End

	ADD R1, LOCA
1	PC_{out}, MAR_{in}, Read, Select4, Add, Z_{in}
2	Z_{out}, PC_{in}, WMFC
3	MDR_{out}, IR_{in}
4	Address field of IRout, MAR_{in}, Read
5	$R1_{out}$, Y_{in}, WMFC
6	MDR_{out}, SelectY, Add, Z_{in}
7	Z_{out}, $R1_{in}$, End

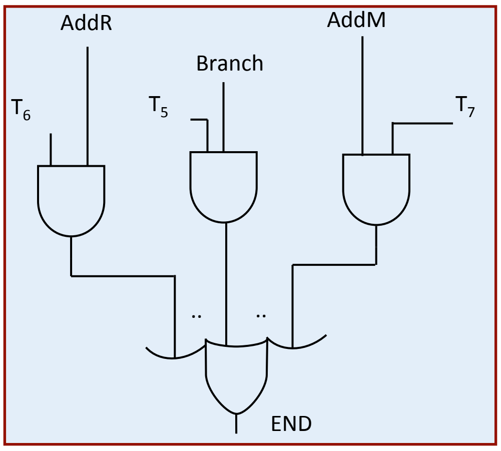
	BRANCH Label
1	PC_{out}, MAR_{in}, Read, Select4, Add, Z_{in}
2	Z_{out}, PC_{in}, Y_{in}, WMFC
3	MDR_{out}, IR_{in}
4	Offset-field-of-IR_{out}, SelectY, Add, Z_{in}
5	Z_{out}, PC_{in}, End

Generation of PC_{in} and END

$PC_{in} = T_2 + T_5.Branch$



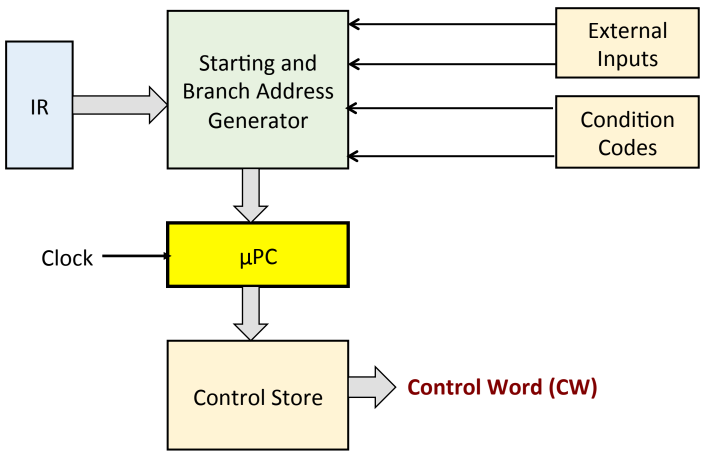
$END = T_6.ADDR + T_5.Branch + T_7.AddM$



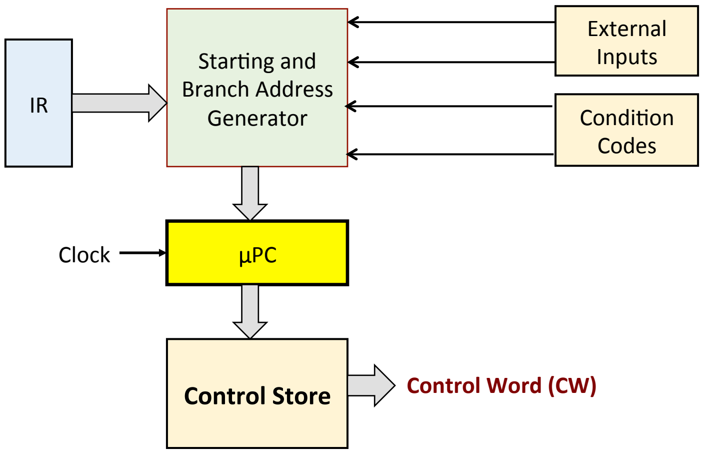
- Using similar logic circuits, all the control signals can be generated.

Microprogrammed Control Unit Design

- Control signals are generated by a program similar to machine language program.
- A *Control Store* (CS) stores the microroutines for all instructions of an ISA.
- The sequence of steps corresponding to the control sequence of a machine instruction is the *microroutine*.
- Each sequence of steps is a *control word* (CW) whose individual bits represent the various control signals.
- Individual control words in a microroutine are called *microinstructions*.



- Control-unit generates the control signals for an instruction by sequentially reading CWs of corresponding microroutine from CS.
- The *μPC* is used to read CWs sequentially from CS.
- Every time a new instruction is loaded into IR, output of *Starting Address Generator* is loaded into μPC.
- Then, μPC is automatically incremented by clock causing successive microinstructions to be read from *Control Store*.



Control Store Contents for an Example Instruction

	ADD R1, R2
1	PC_{out}, MAR_{in}, Read, Select4, Add, Z_{in}
2	Z_{out}, PC_{in}, WMFC
3	MDR_{out}, IR_{in}
4	$R1_{out}$, Y_{in}
5	$R2_{out}$, SelectY, Add, Z_{in}
6	Z_{out}, $R1_{in}$, End

Control Store for "ADD R1, R2"

Micro-instr.	…	PC_{in}	PC_{out}	MAR_{in}	Read	MDR_{out}	IR_{in}	Y_{in}	Select4	SelectY	Add	Z_{in}	Z_{out}	$R1_{out}$	$R1_{in}$	$R2_{out}$	WMFC	End	…
1	0	0	1	1	1	0	0	0	1	0	1	1	0	0	0	0	0	0	0
2	0	1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	0	0
3	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0	1	1	1	0	0	0	1	0	0	0
6	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	1	0

Control Store Contents for Another Instruction

	BRANCH Label
1	PC_{out}, MAR_{in}, Read, Select4, Add, Z_{in}
2	Z_{out}, PC_{in}, Y_{in}, WMFC
3	MDR_{out}, IR_{in}
4	Offset-field-of-IR_{out}, SelectY, Add, Z_{in}
5	Z_{out}, PC_{in}, End

Control Store for "BRANCH Label"

Micro-instr.	…	PC_{in}	PC_{out}	MAR_{in}	Read	MDR_{out}	IR_{in}	Y_{in}	Select4	SelectY	Add	Z_{in}	Z_{out}	IR_{out}	WMFC	End	…
1	0	0	1	1	1	0	0	0	1	0	1	1	0	0	0	0	0
2	0	1	0	0	0	0	0	1	0	0	0	0	1	0	1	0	0
3	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	1	1	1	0	1	0	0	0
5	0	1	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0

Using Conditional Branching

- The previous organization cannot handle the situation when the control unit is required to check the status of the condition codes or external inputs to choose between alternative courses of action.
- Possible solution:
  - Use conditional branch *microinstruction*.

BZ Label

Address	Microinstruction
0	PC_{out}, MAR_{in}, Read, Select4, Add, Z_{in}
1	Z_{out}, PC_{in}, Y_{in}, WMFC
2	MDR_{out}, IR_{in}
3	Branch to starting address of appropriate microroutine
....	....
25	If Z=0, then branch to microinstruction 0
26	Offset-field-of-IR_{out}, SelectY, Add, Z_{in}
27	Z_{out}, PC_{in}, End

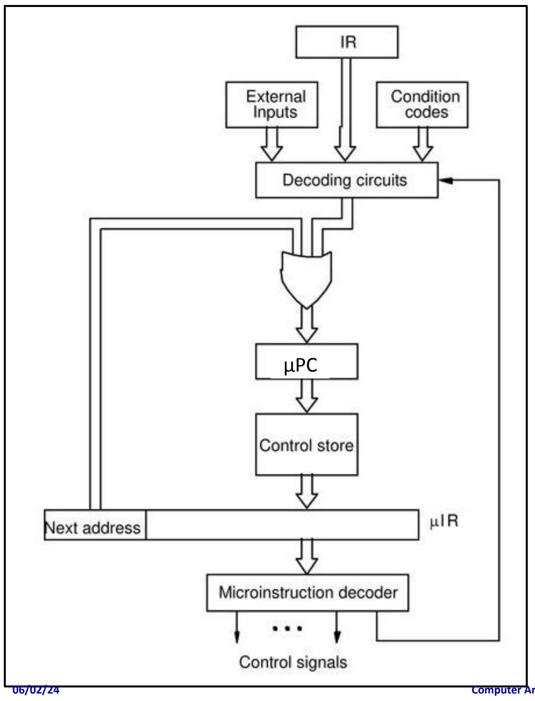
Microprogram Sequencing

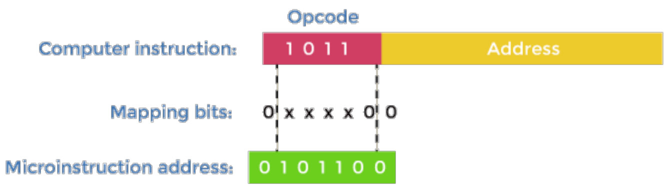
- If all microprograms require only straightforward sequential execution of microinstructions except for branches, letting a µPC govern the sequencing would be efficient.
- However, two disadvantages:
  - Having a separate microroutine for each machine instruction results in a large total number of microinstructions and a large control store.
  - Longer execution time because it takes more time to carry out the required branches.

Microinstructions with Next-Address Field

- The microprogram requires several branch microinstructions, which perform no useful operation in the datapath.
- An alternative approach:
  - Include an address field as a part of every microinstruction to indicate the location of the next microinstruction to be fetched.

- Pros: separate branch microinstructions are virtually eliminated.
- Cons: additional bits required for the address field.





Mapping from instruction code to microinstruction address

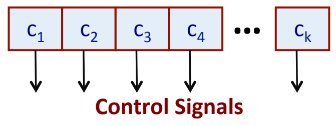
Horizontal versus Vertical Microinstruction Encoding

- Broadly there are two alternate schemes to code the control signals in the control memory.
  a) **Horizontal Micro-instruction Encoding**
    - Each control signal is represented by a bit in the micro-instruction.
    - Fewer control store words, with more bits per word.

  b) **Vertical Micro-instruction Encoding**
    - Each control word represents a single micro-instruction in encoded form.
    - k-bit control word can support up to 2^k micro-instructions.
    - More control store words, with fewer bits per word.

- There can be a tradeoff between horizontal and vertical micro-instruction encoding.
  - Sometimes referred to as **Diagonal Micro-instruction Encoding**.
  - The control signals are grouped into sets S_1, S_2, etc., such that the control signals within a set are mutually exclusive.
- Summary:
  - Horizontal encoding supports unlimited parallelism among micro-instructions.
  - Vertical encoding supports strictly sequential execution of micro-instructions.
  - Diagonal encoding does not sacrifice the required level of parallelism, but uses less number of bits per control word as compared to horizontal encoding.

(a) Horizontal Micro-instruction Encoding



- Suppose that there are *k* control signals: $c_1, c_2, \ldots, c_k$.
- In horizontal encoding, every control word stored in control memory (CM) consists of *k* bits, one bit for every control signal.
- Several bits in a control word can be 1:
  - Parallel activation of several micro-operations in a single time step.

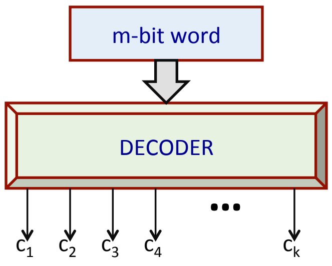
0	1	0	1	1	0

→ c_2, c_4 and c_5 are activated together

- Advantage:
  - Unlimited parallelism is possible in the activation of the micro-operations.
- Disadvantage:
  - Size of the control memory is large (word size is much longer).
  - Cost of implementation is higher.

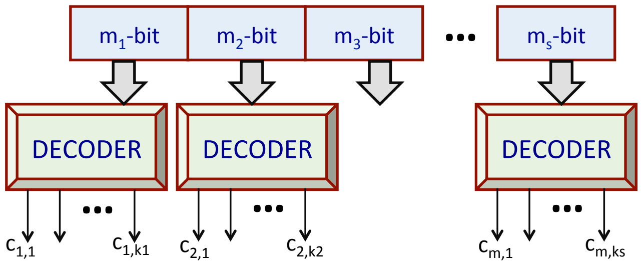
(b) Vertical Micro-instruction Encoding

- Again consider that there are k control signals: c_1, c_2, …, c_k.
- We encode the control signals in an m-bit word in the control memory, where $k \le 2^m$.
- Depending on the m-bit control word, exactly one control signal will be activated (= 1), while all others will remain de-activated (= 0).
  - At most one control signal can be activated in a time step.



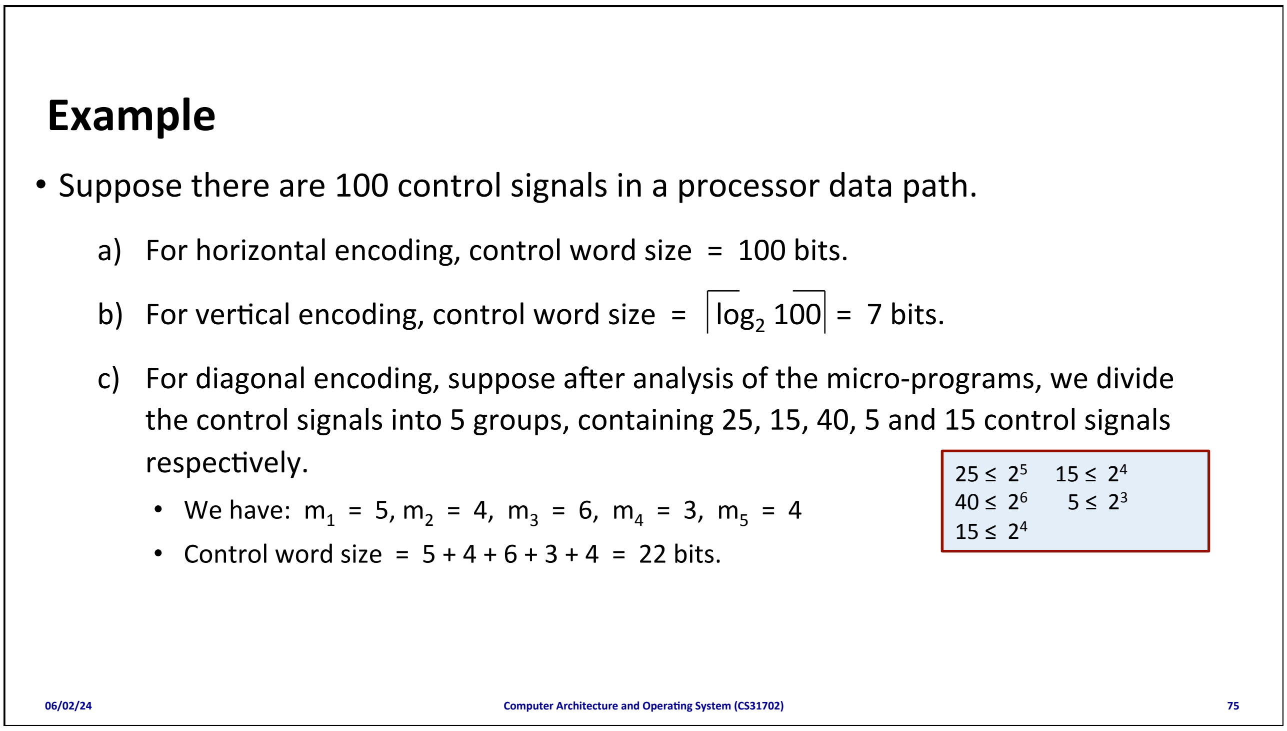
- Advantage:
  - Requires much smaller word size in control memory.
  - Low cost of implementation.
- Disadvantage:
  - More than one control signals cannot be activated at a time.
  - Requires sequential activation of the control signals, and hence more number of time steps.

(c) Diagonal Micro-instruction Encoding



- Suppose we group the set of *k* control signals into *s* groups, containing $k_1, k_2, \ldots, k_s$ signals.
- We encode the control signals in groups as shown, where $k_i \leq 2^{mi}$.
    - Within a group, at most one control signal can be activated in a time step.
    - Parallelism across groups is allowed.

- Advantages:
    - Maximum parallelism as required by the micro-programs can be supported.
    - Word size of control memory is less than that for horizontal encoding.
    - Used in practice.
- Disadvantages:
    - Multiple decoders (though smaller in sizes) are required.



Example

- Suppose there are 100 control signals in a processor data path.
  a) For horizontal encoding, control word size = 100 bits.
  b) For vertical encoding, control word size = $\lceil \log_2 100 \rceil$ = 7 bits.
  c) For diagonal encoding, suppose after analysis of the micro-programs, we divide the control signals into 5 groups, containing 25, 15, 40, 5 and 15 control signals respectively.
    - We have: $m_1 = 5, m_2 = 4, m_3 = 6, m_4 = 3, m_5 = 4$
    - Control word size = 5 + 4 + 6 + 3 + 4 = 22 bits.

$25 \leq 2^5$ $15 \leq 2^4$
$40 \leq 2^6$ $5 \leq 2^3$
$15 \leq 2^4$