Computer Organization and Architecture

Module 4

Design of Control Unit

Prof. Indranil Sengupta

Dr. Sarani Bhattacharya

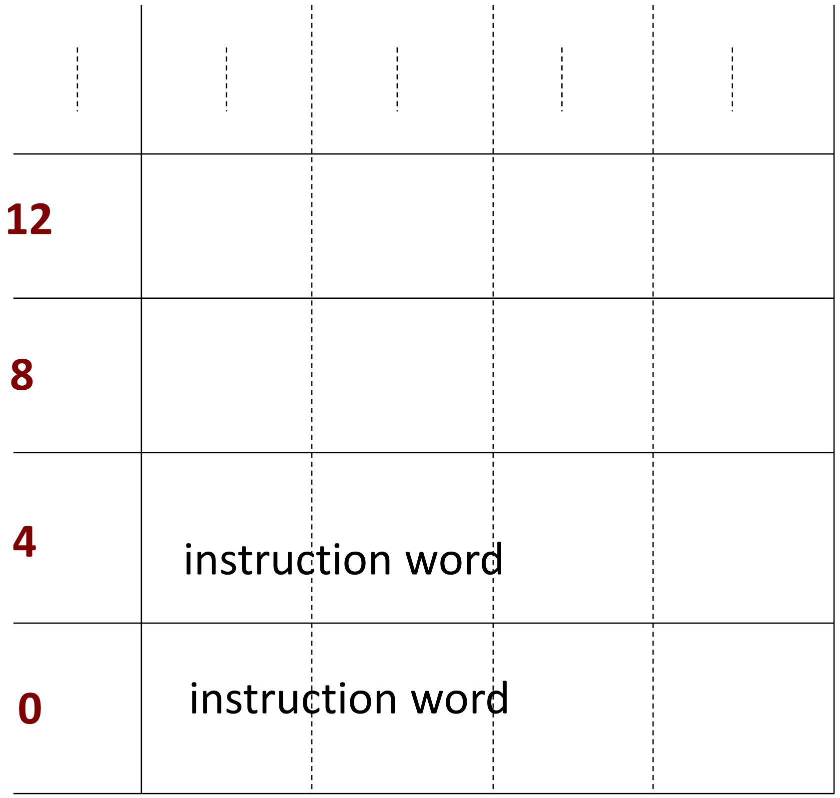
Department of Computer Science and Engineering

IIT Kharagpur

Design of Control Unit

Instructions

- Instructions are stored in main memory.
- Program Counter (*PC*) points to the next instruction.
    - MIPS32 instructions are 4 bytes (32 bits) long.
    - All instructions starts from an address that is multiple of 4 (last 2 bits 00).
    - Normally, *PC* is incremented by 4 to point to the next instruction.
    - For branch, *PC* is loaded with the address of the target instruction.

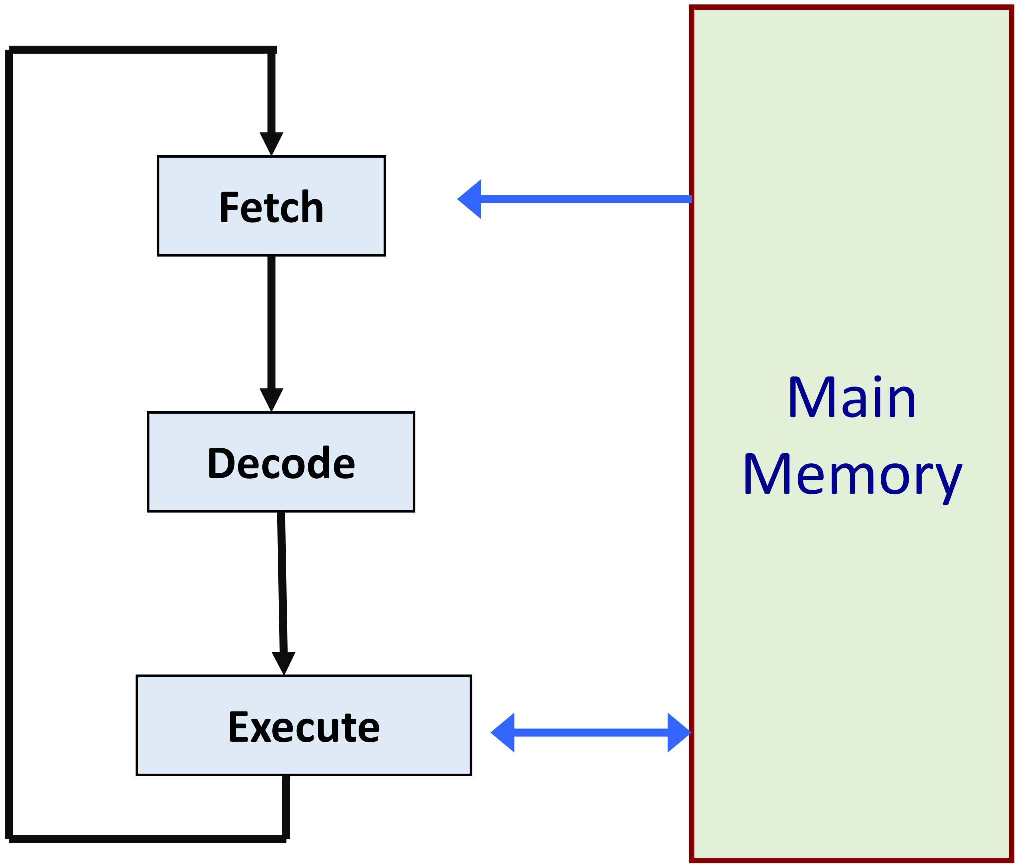


Addressing a Byte in Memory

- Each byte in memory has a unique address.
  - Memory is said to be *byte addressable*.
- Typically the instructions are of 4 bytes, hence the instruction memory is addressed in terms of 4 bytes (word length = 32 bits).
- When an instruction is executed, PC is *incremented by 4* to point to the next instruction.
  - In MIPS32, words are byte aligned.
  - Every word (including instruction) starts from a memory address that is some multiple of 4 (i.e., last two bits are `00').

How an instruction Gets Executed?

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



The Fetch-Execute Cycle

1) Fetch the next instruction from memory.
2) Decode the instruction.
3) Execute the operation.
   - Get data from memory if needed (data not available in the processor).
   - Perform the required operation on the data.
   - May also store the result back in memory or register.

Registers: PC and IR

- **Program Counter (PC)** holds the address of the memory location containing the next instruction to be executed.
- **Instruction Register (IR)** contains the current instruction being executed.
- Basic processing cycle:
  - Instruction Fetch (IF)

    $IR \leftarrow Mem\,[PC]$
  - Considering the word length of the machine is 32 bits, the PC is incremented by 4 to point to the next instruction.

    $PC \leftarrow PC + 4$
  - Carry out the operations specified in *IR*.

Example: Add R1, R2

Address	Instruction
1000	ADD R1, R2
1004	MUL R3, R4

a) PC = 1000

b) MAR = 1000

c) PC = PC + 4 = 1004

d) MDR = “ADD R1, R2”

e) IR = “ADD R1, R2”

(*Decode and finally execute*)

f) R1 = R1 + R2

May require one or more steps depending on the target architecture.

Requirement for Instruction Execution

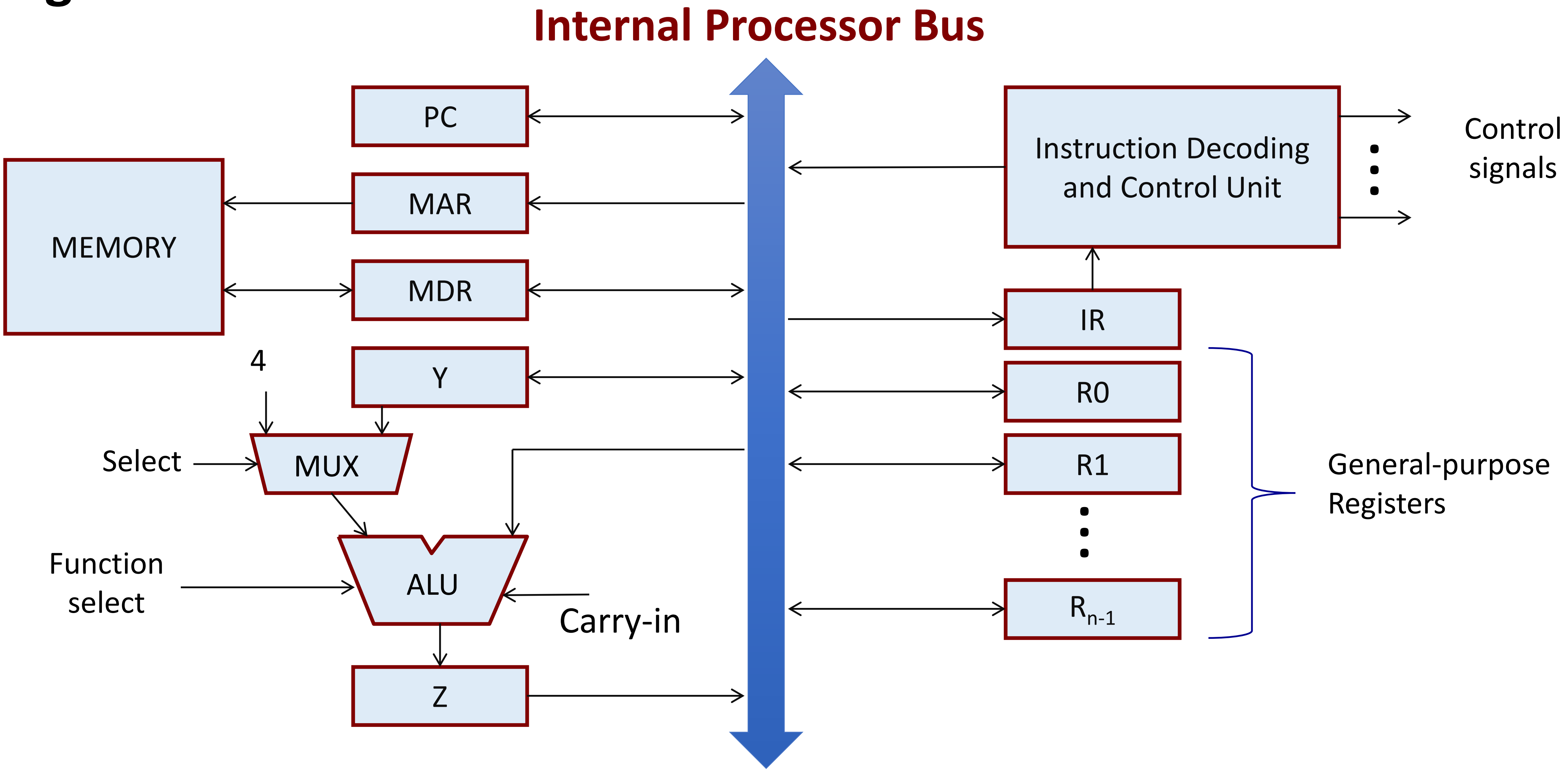
- The necessary registers must be present.
- The internal organization of the registers must be known.
- The data path must be known.
- For instruction execution, a number of *micro-operations* are carried out on the data path.
  - An instruction consists of several micro-operations or micro-instructions.
  - Typically involves movement of data.

Kinds of Data Movement

- Broadly three types:
  a) Register to Register
  b) Register to ALU
  c) ALU to Register

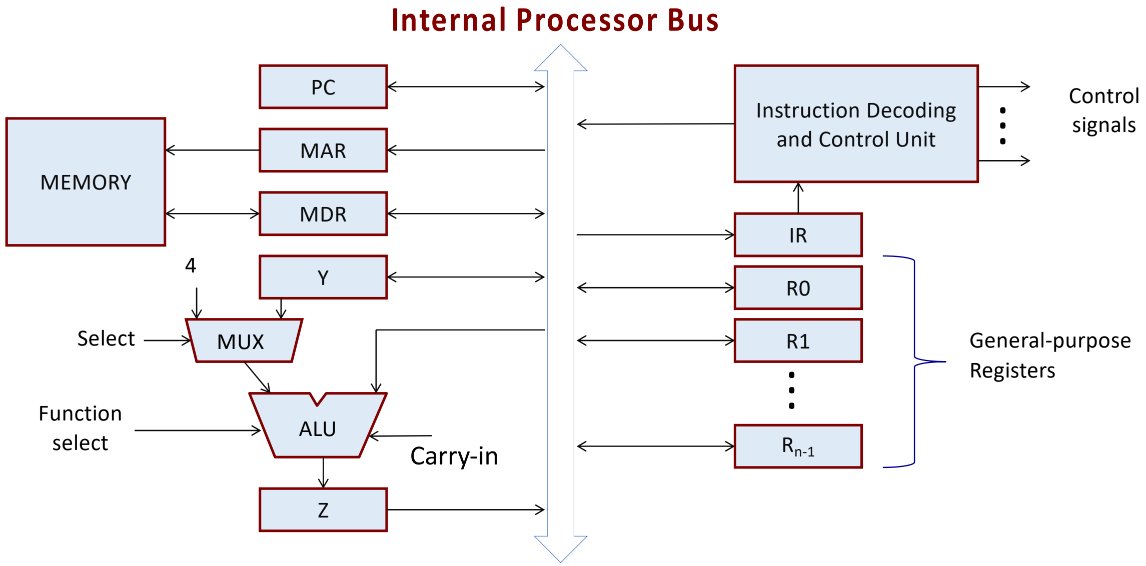
- Data movement is supported in the data path by:
  - The Registers
  - The Bus (single or multiple)
  - The ALU temporary Register (*Y* and *Z*)

Single Bus Organization

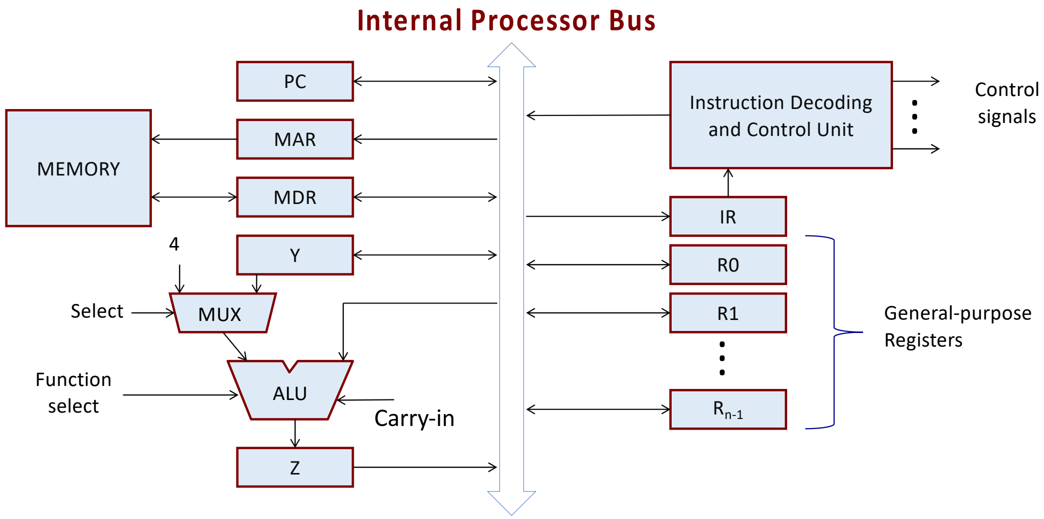


Single Internal Bus Organization

- 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.
- Registers *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 *data path*.
- The control unit that generates the control signals in proper sequence is referred to as the *control path*.



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]

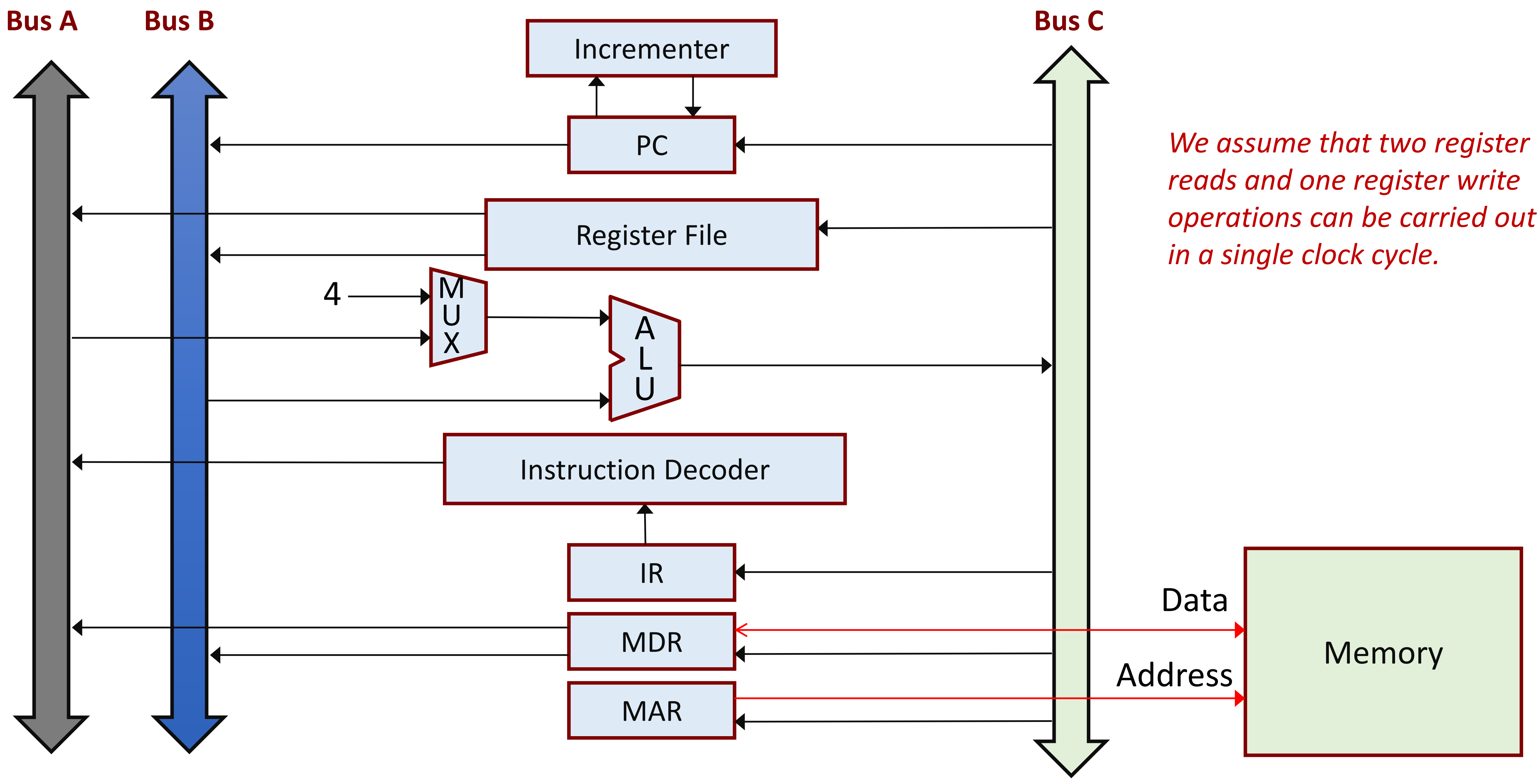
- Store a word of data from a register into a given memory location.

  STOR 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.

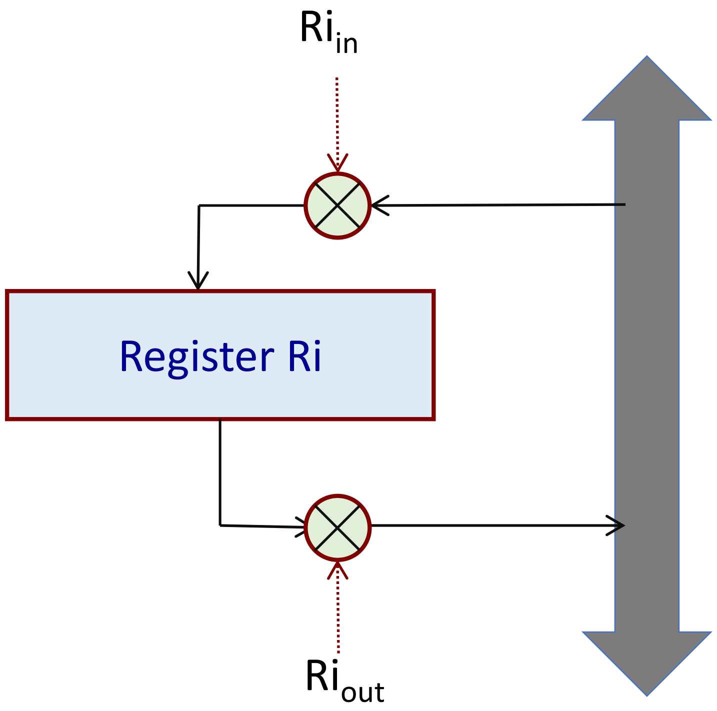
- Less number of cycles required to execute an instruction compared to single bus organization.



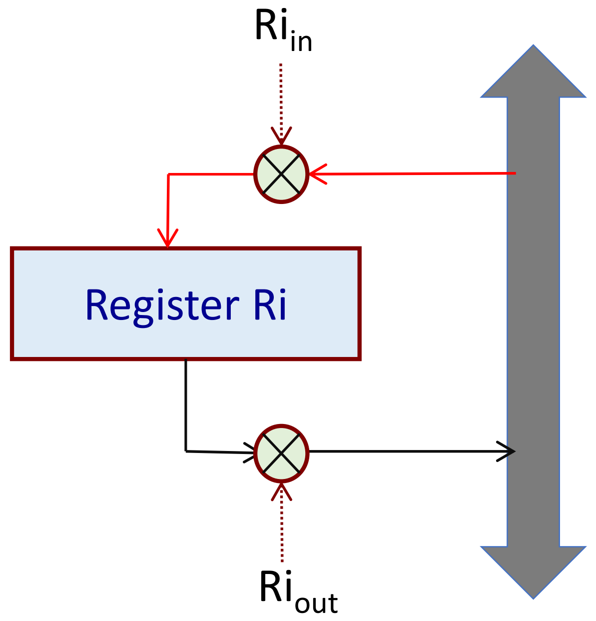
We assume that two register reads and one register write operations can be carried out in a single clock cycle.

Organization of a Register

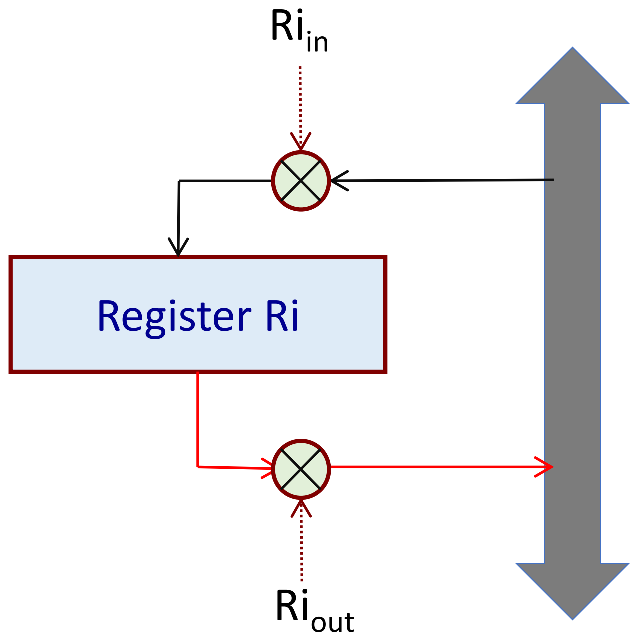
- A register is used for temporary storage of data (parallel-in, parallel-out, etc.).
- A register Ri typically has two control signals.
  - Ri_{in} : used to load the register with data from the bus.
  - Ri_{out} : used to place the data stored in the register on the bus.
- Input and output lines of the register Ri are connected to the bus via controlled switches.
  - If Ri_{out} is not selected, the register outputs are set in the high impedance state.



- 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.

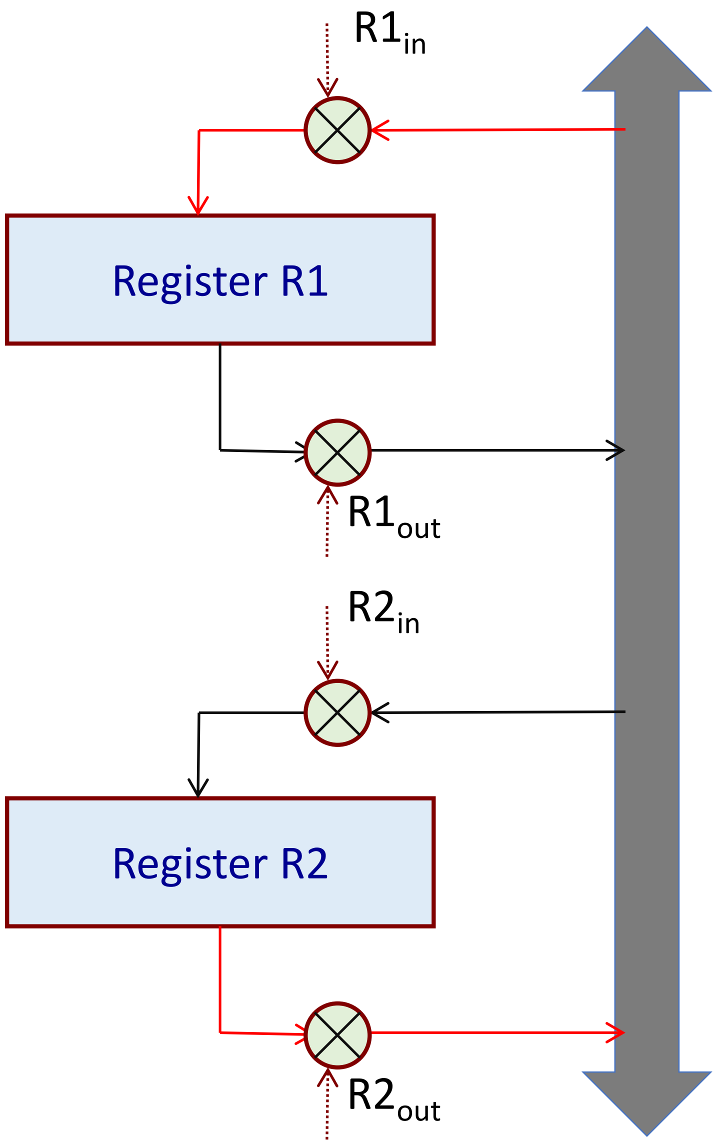


Register Transfer

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.

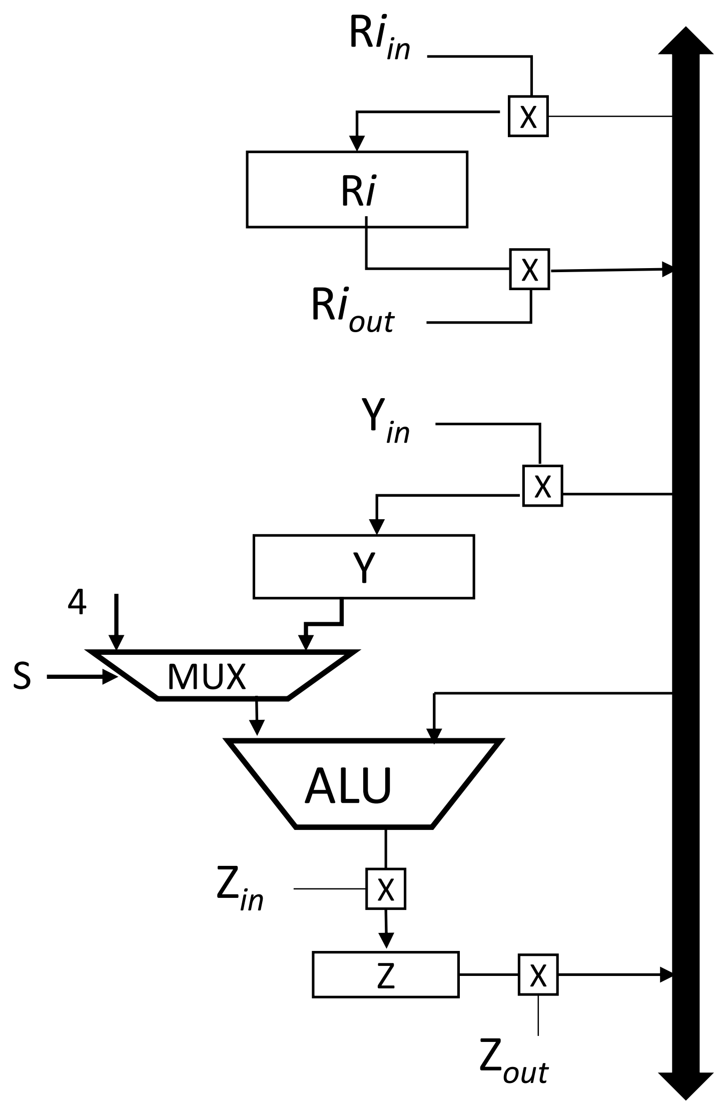
  One through Y (R1) and another (R2) directly from internal bus.

- 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}, $R1_{in}$



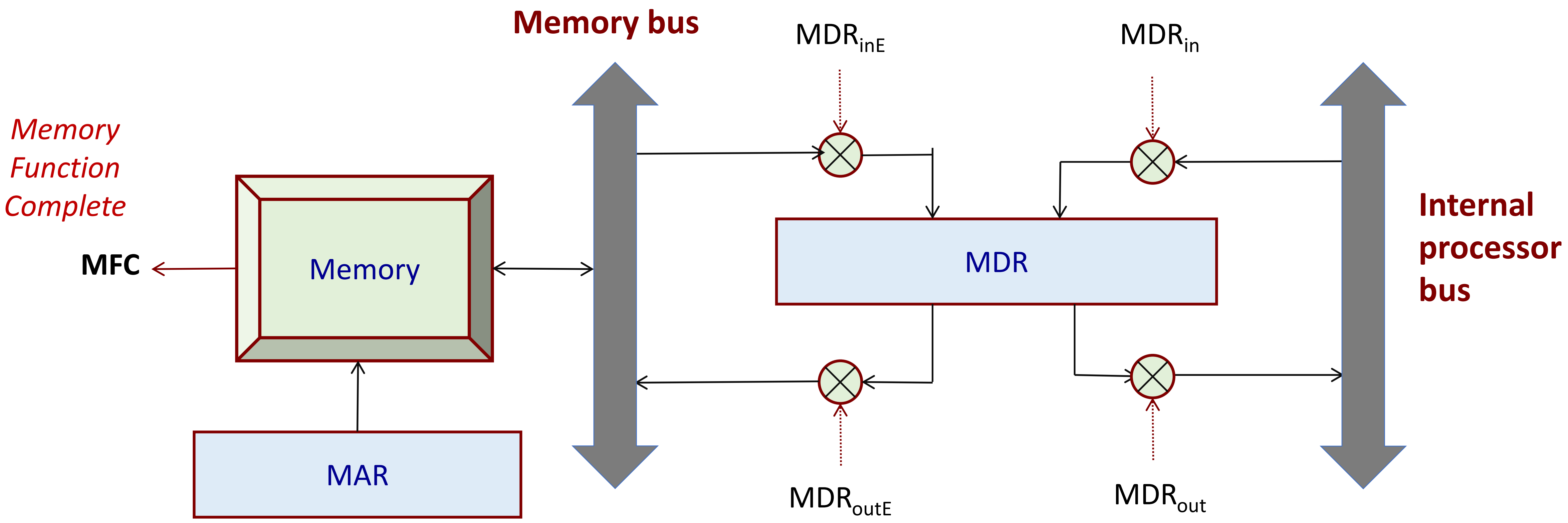
Fetching a Word from Memory

- The steps involved to fetch a word from memory:
  - The processor specifies the address of the memory location where the data or instruction is stored (move to *MAR*).
  - 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 is transferred to the required register or ALU for further operation.

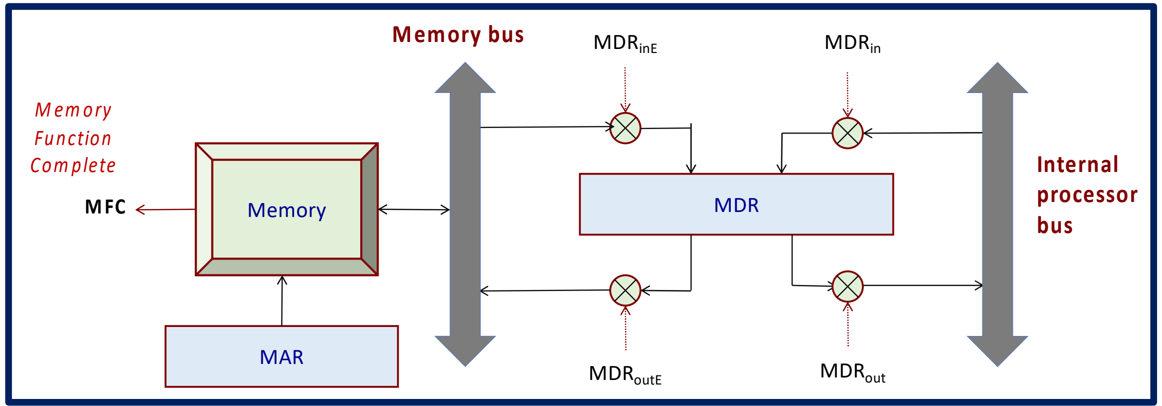
Storing a Word into Memory

- The steps involved to store a word into the memory:
  - The processor specifies the address of the memory location where the data is to be written (move to *MAR*).
  - 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 process 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 contents 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)

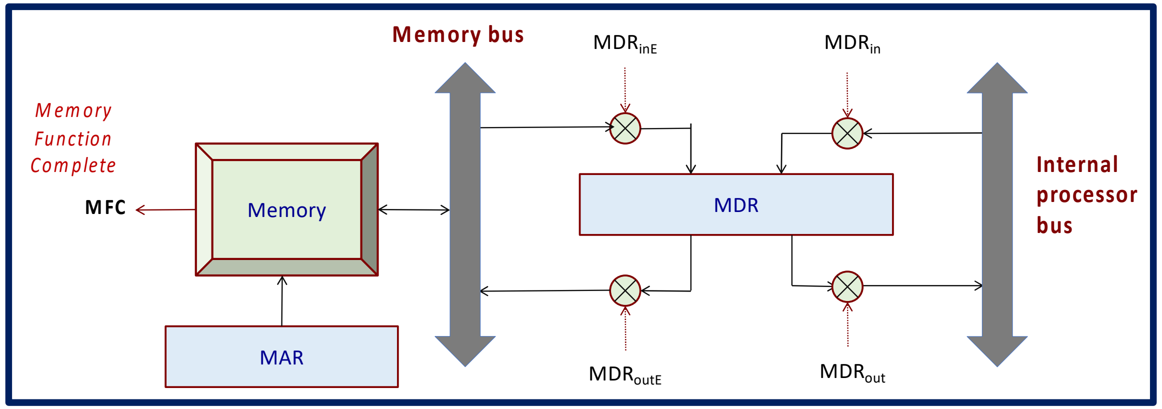
1. MAR ← R2
2. Start a Read operation on the memory bus
3. Wait for the MFC response from the memory
4. Load MDR from the memory
5. R1 ← MDR

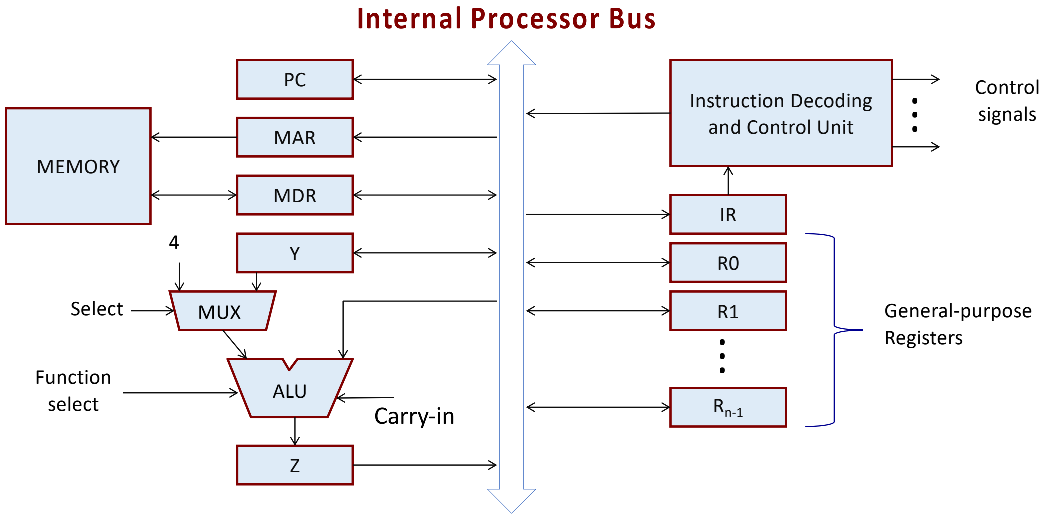
Control steps:

a) $R2_{out}$, MAR_{in}, Read

b) MDR_{inE}, WMFC

c) MDR_{out}, $R1_{in}$





Store a word: MOVE (R1), R2

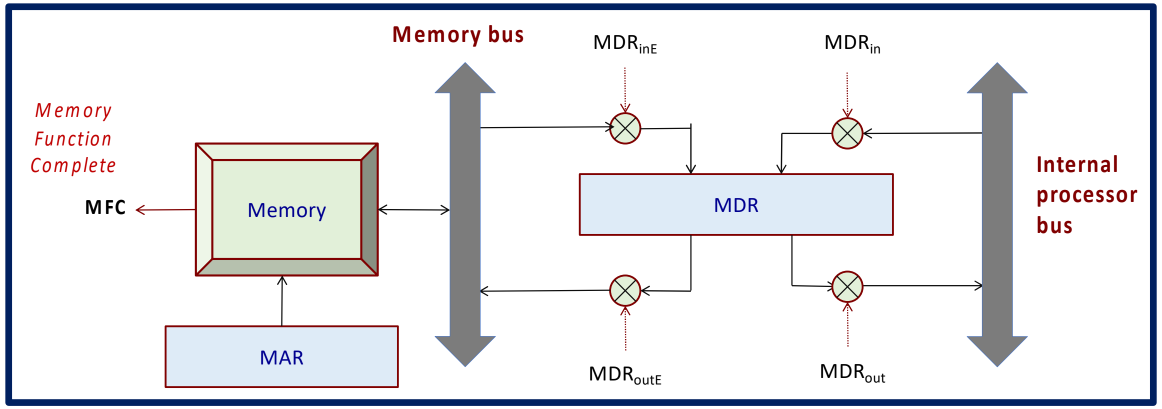
1. MAR ← R1
2. MDR ← R2
3. Start a Write operation on the memory bus
4. Wait for the MFC response from the memory

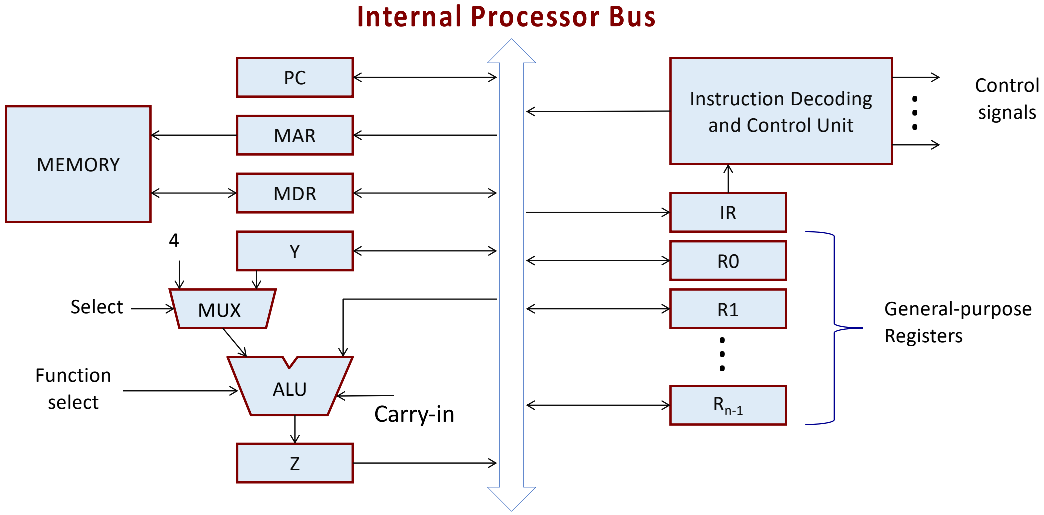
Control steps:

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}

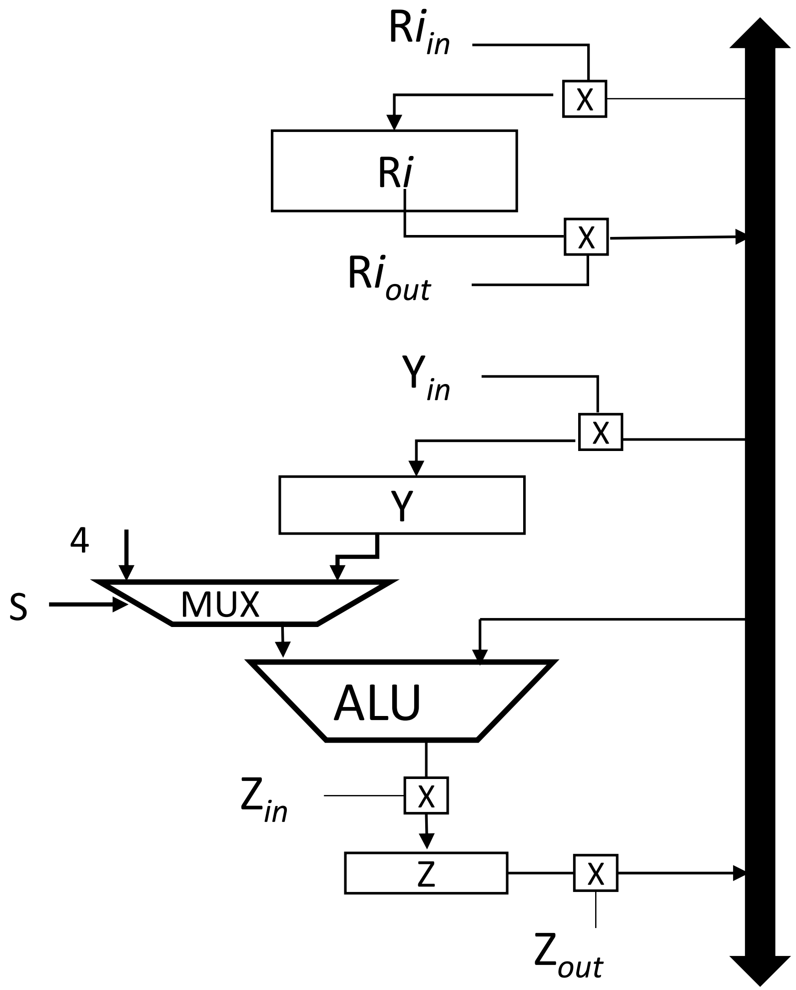
T2: Z_{out}, PC_{in}, Y_{in}, WMFC

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

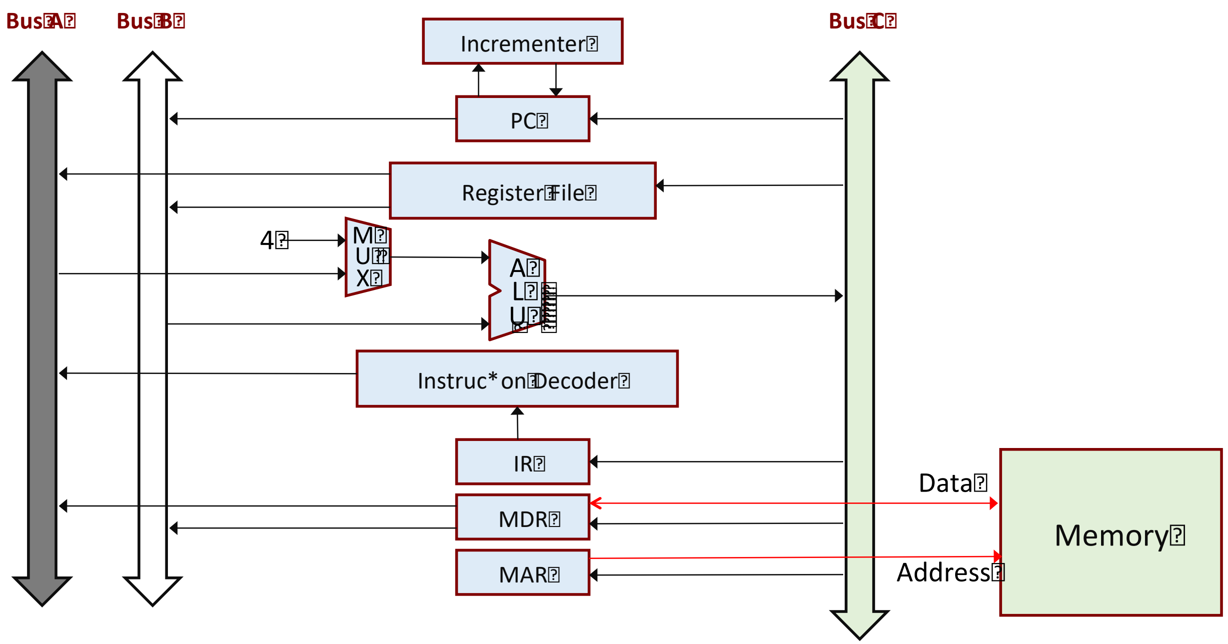
T1: PC_{out}, R = B, MAR_{in} , READ, IncPC

T2: WMFC

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

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

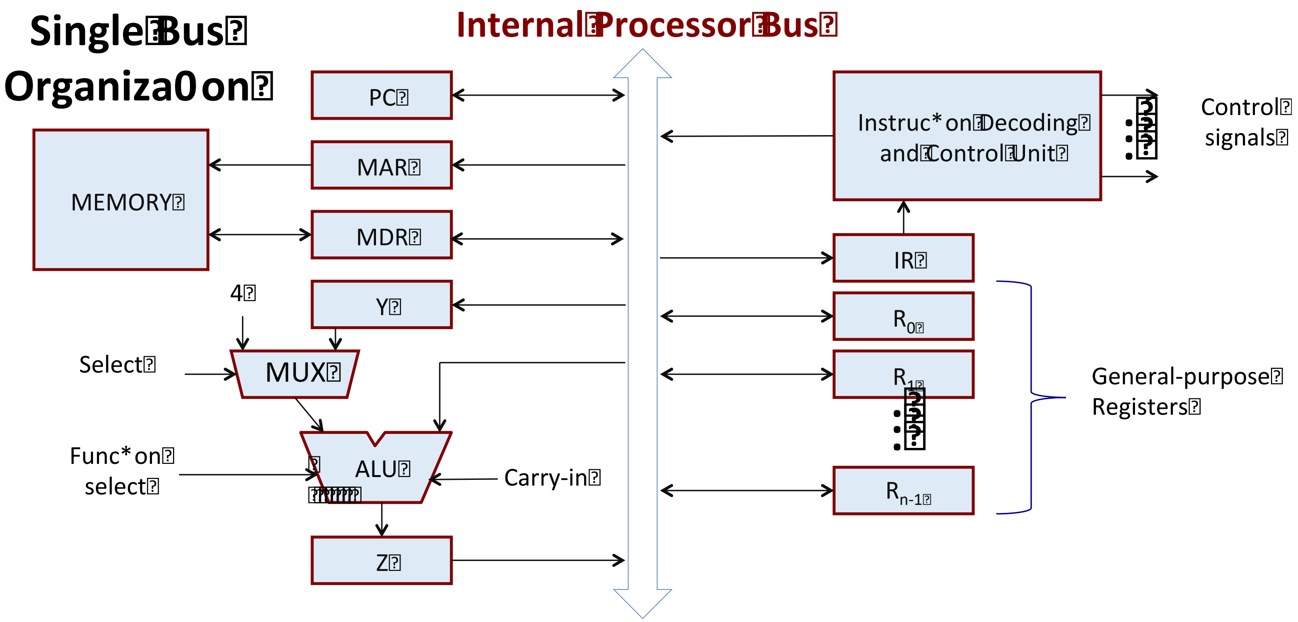
R = B means that the ALU function is selected such that data on Bus-B is transferred to the ALU output (i.e., Bus-C).



Micro-operations Examples

Introduction

- We select a set of 12 instructions.
- Discuss the control signals required to execute these instructions on the single-bus processor architecture.



The set of 12 instructions chosen

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
12.	HALT		// Machine Halt

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

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

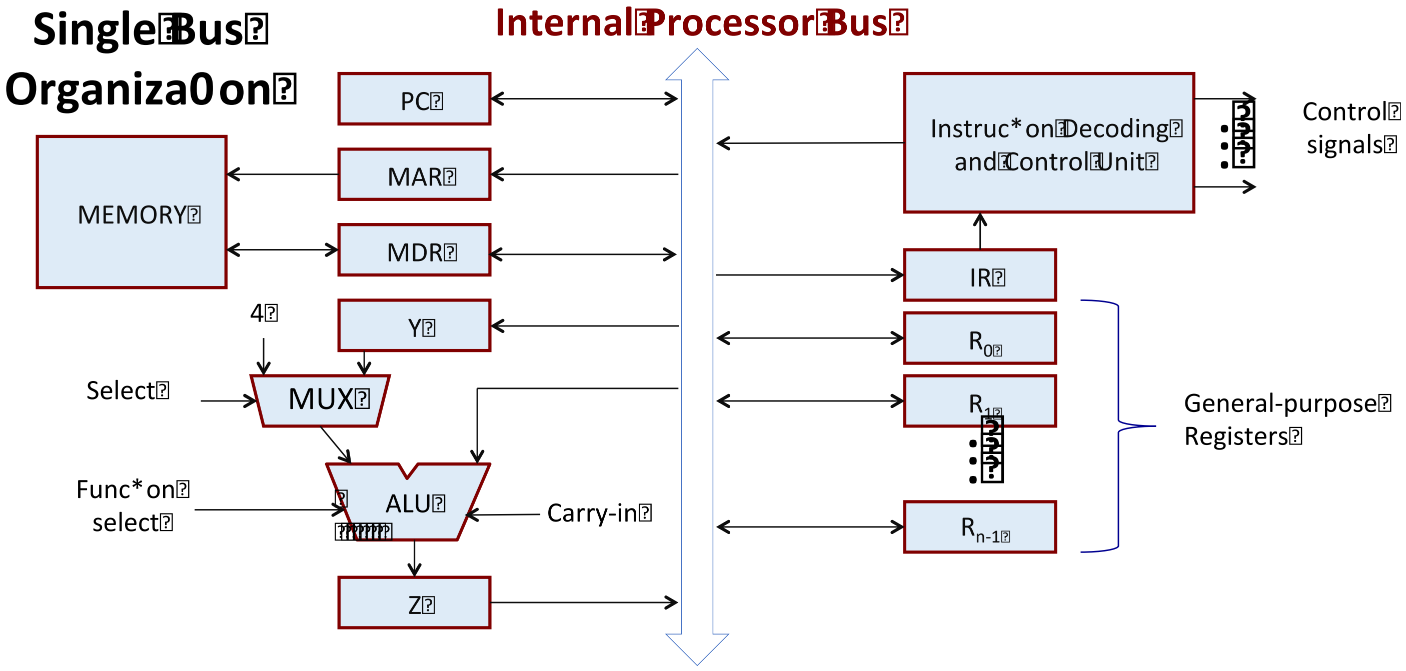
T2: Z_{out}, PC_{in}, Y_{in}, WMFC

T3: MDR_{out}, IR_{in}

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

T5: $R2_{out}$, SelectY, Add, Z_{in}

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



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

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

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

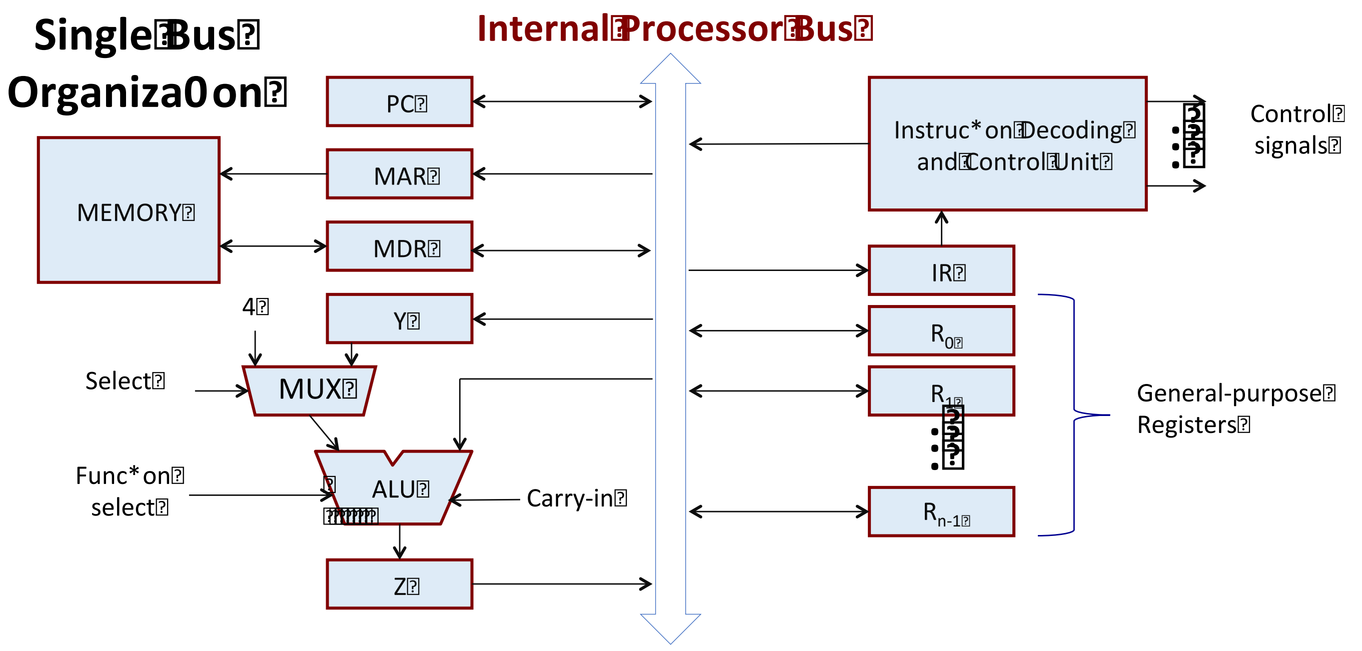
T3: MDR_{out}, IR_{in}

T4: Address field of IR_{out}, MAR_{in}, Read

T5: $R1_{out}$, Y_{in}, WMFC

T6: MDR_{out}, SelectY, Add, Z_{in}

T7: Z_{out}, $R1_{in}$, End



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

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

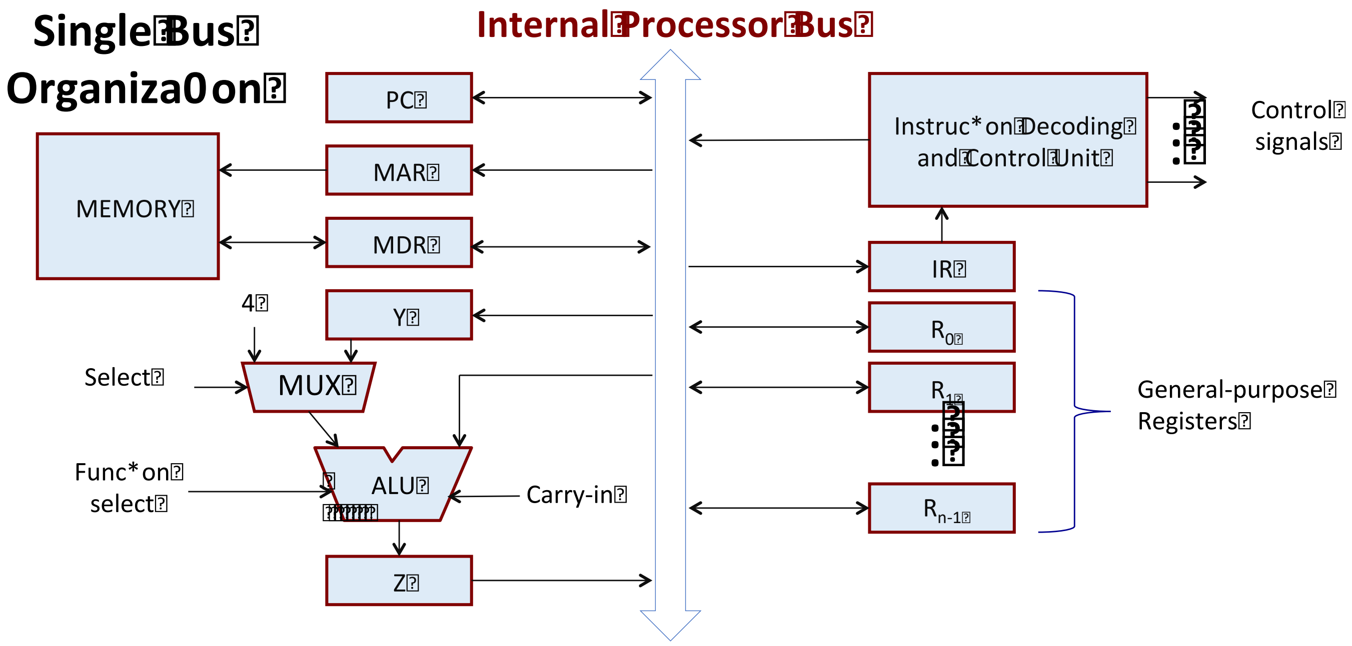
T2: Z_{out}, PC_{in}, Y_{in}, WMFC

T3: MDR_{out}, IR_{in}

T4: Address field of IR_{out}, MAR_{in}, Read

T5: WMFC

T6: MDR_{out}, $R1_{in}$, End



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

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

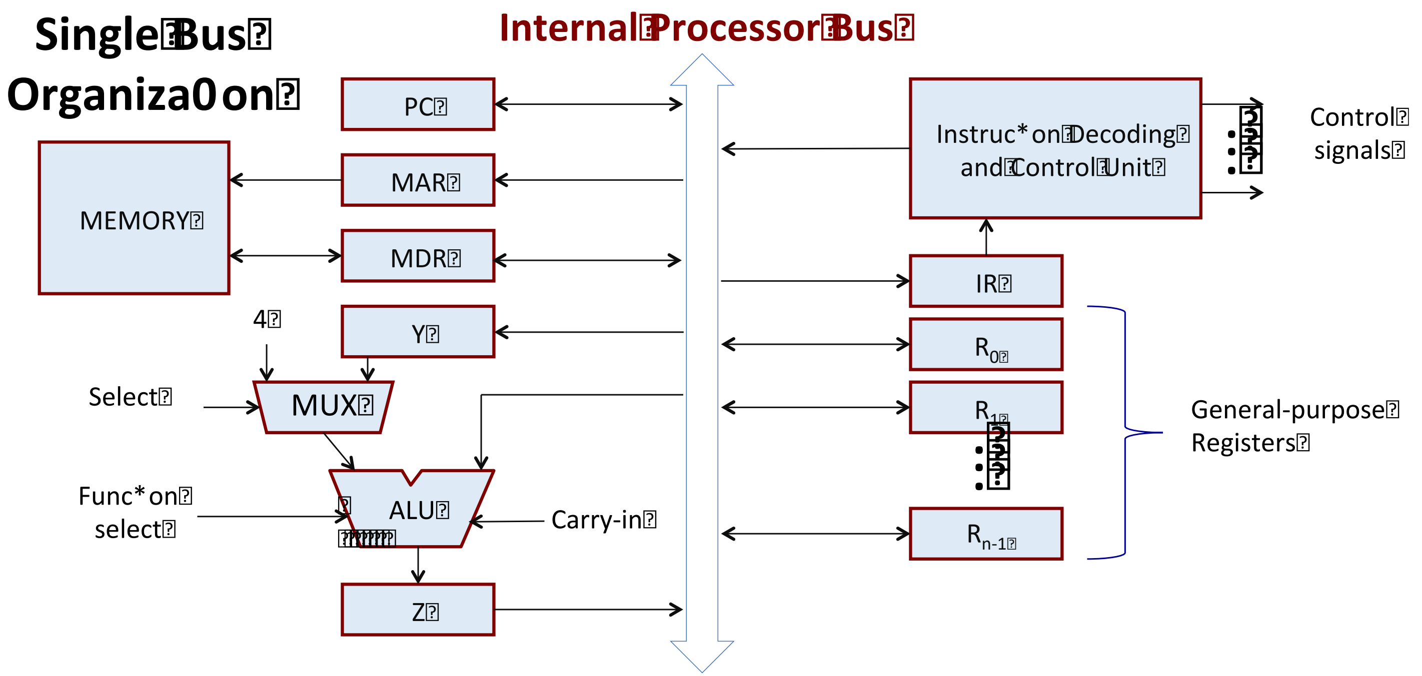
T2: Z_{out}, PC_{in}, Y_{in}, WMFC

T3: MDR_{out}, IR_{in}

T4: Address field of IR_{out}, MAR_{in}

T5: $R1_{out}$, MDR_{in}, Write

T6: MDR_{outE}, WMFC, End



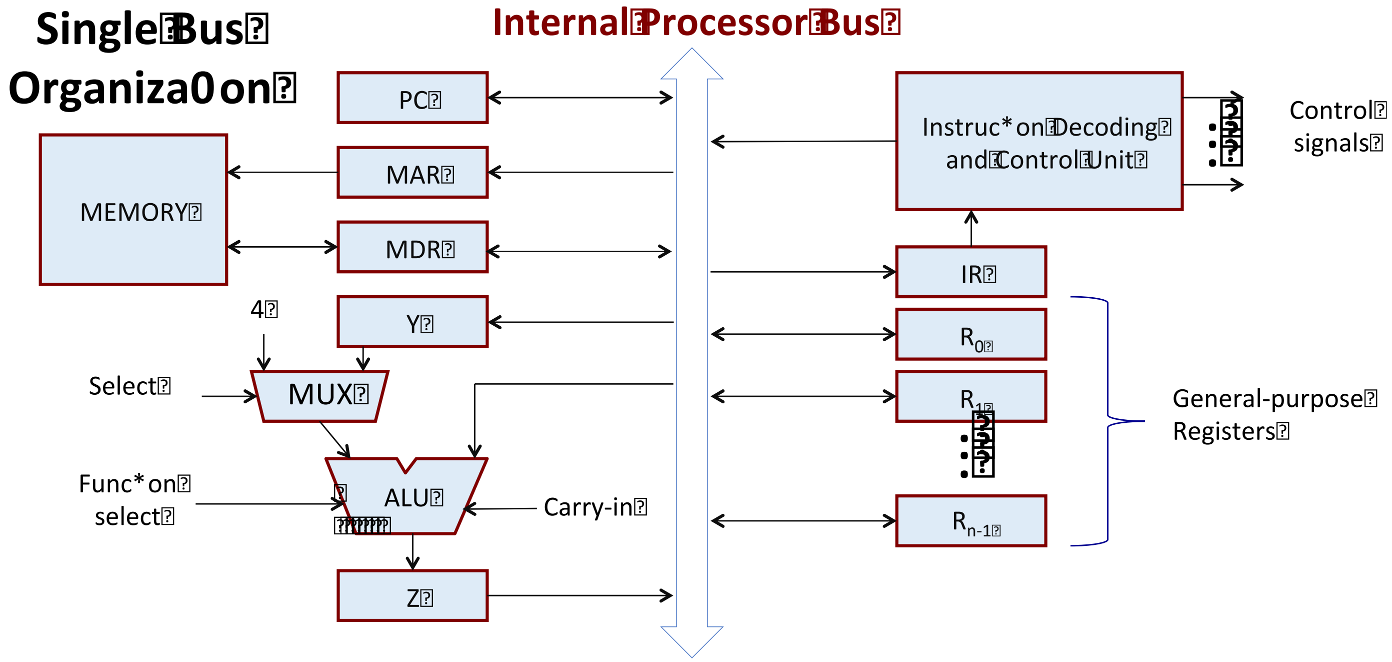
5. MOVE R1, R2 (R1 = R2)

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

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

T3: MDR_{out}, IR_{in}

T4: $R2_{out}$, $R1_{in,}$ END



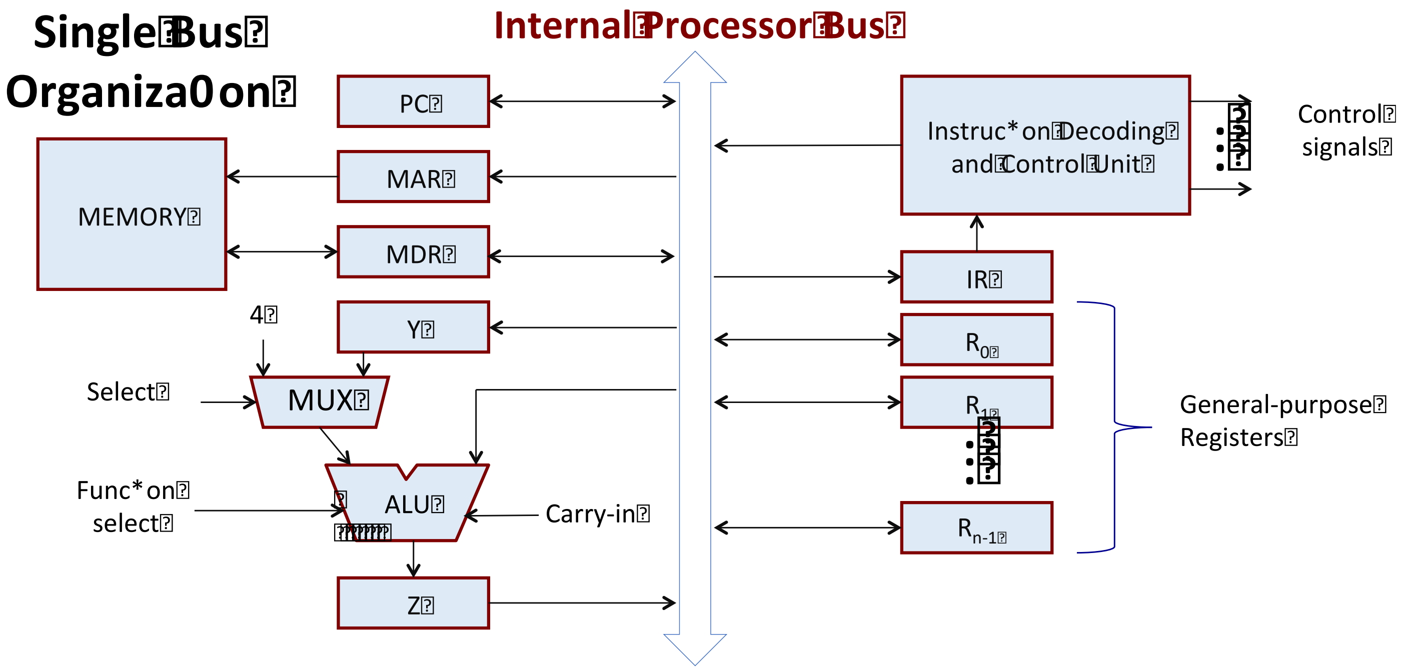
6. MOVE R1, #10 (R1 = 10)

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

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

T3: MDR_{out}, IR_{in}

T4: Immediate field of IR_{out}, $R1_{in}$, END



7. BRANCH Label (PC = PC + offset)

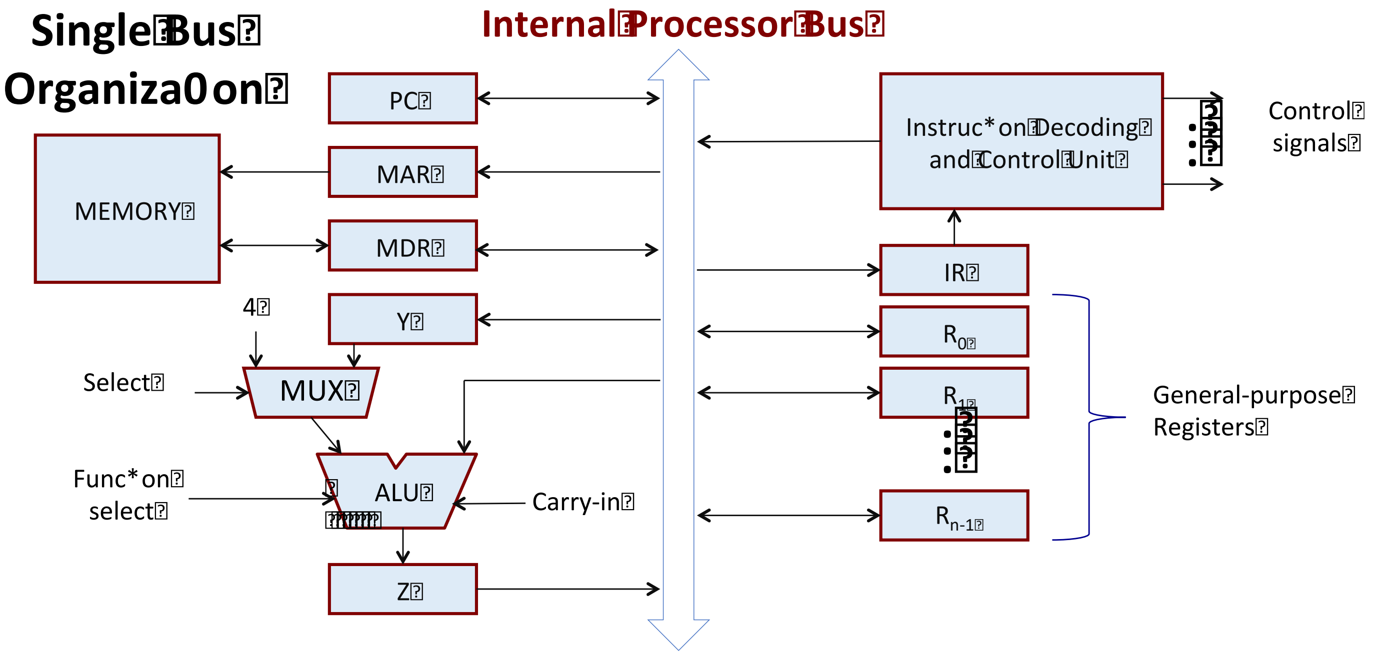
T1: PC_{out}, MAR_{in}, Read, Select4, Add, Z_{in}

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

T3: MDR_{out}, IR_{in}

T4: Offset-field-of-IR_{out}, SelectY, Add, Z_{in}

T5: Z_{out}, PC_{in}, End



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

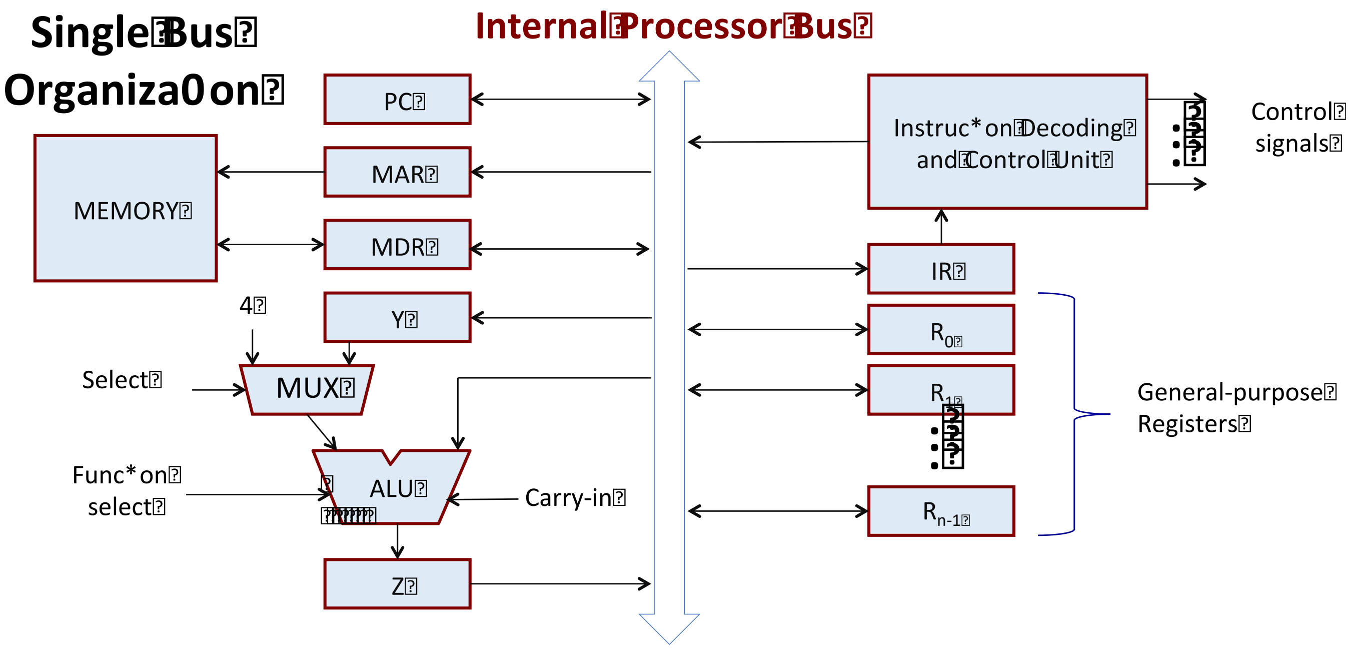
T1: PC_{out}, MAR_{in}, Read, Select4, Add, Z_{in}

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

T3: MDR_{out}, IR_{in}

T4: Offset-field-of-IR_{out}, SelectY, Add, Z_{in}, If Z=0 then End

T5: Z_{out}, PC_{in}, End



9. INC R1 (R1 = R1 + 4)

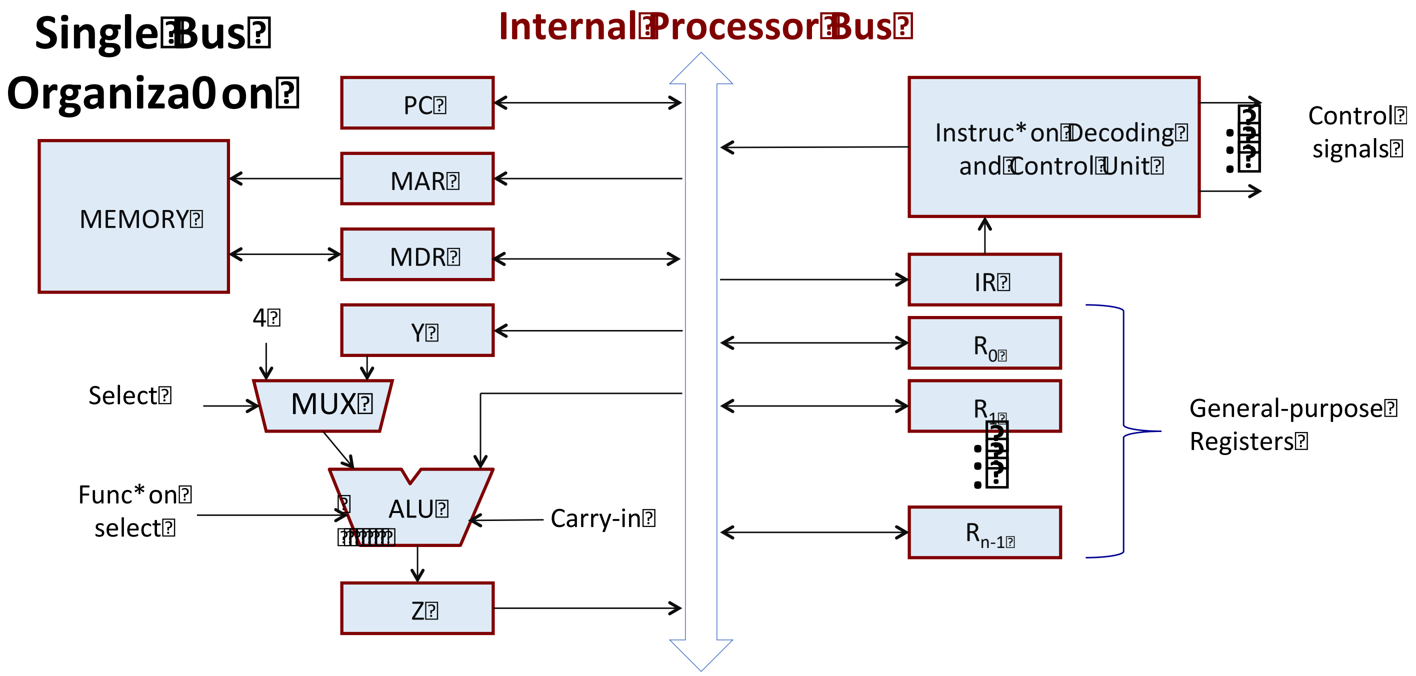
T1: PC_{out}, MAR_{in}, Read, Select4, Add, Z_{in}

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

T3: MDR_{out}, IR_{in}

T4: $R1_{out}$, Select4, Add, Z_{in}

T5: Z_{out}, $R1_{in}$, End



10. DEC R1 (R1 = R1 – 4)

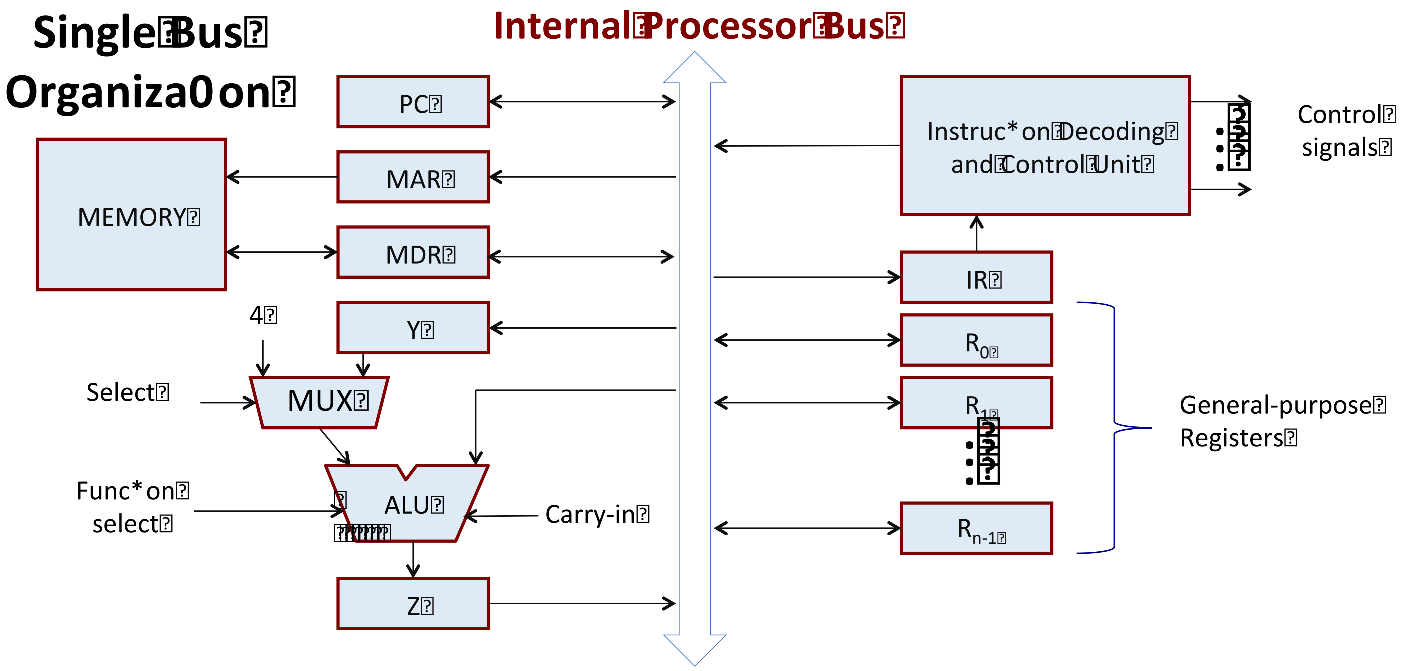
T1: PC_{out}, MAR_{in}, Read, Select4, Add, Z_{in}

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

T3: MDR_{out}, IR_{in}

T4: $R1_{out}$, Select4, Sub, Z_{in}

T5: Z_{out}, $R1_{in}$, End



11. CMP R1, R2

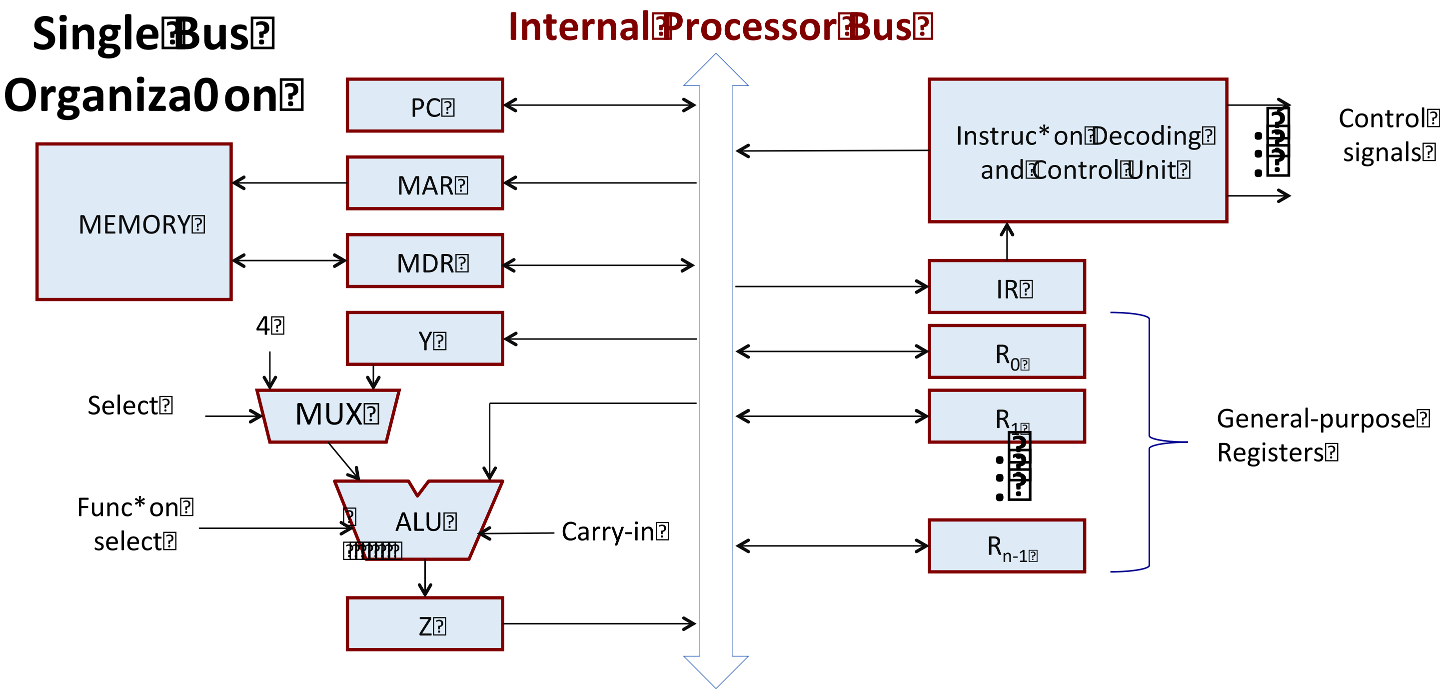
T1: PC_{out}, MAR_{in}, Read, Select4, Add, Z_{in}

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

T3: MDR_{out}, IR_{in}

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

T5: $R2_{out}$, SelectY, Sub, Z_{in}, End



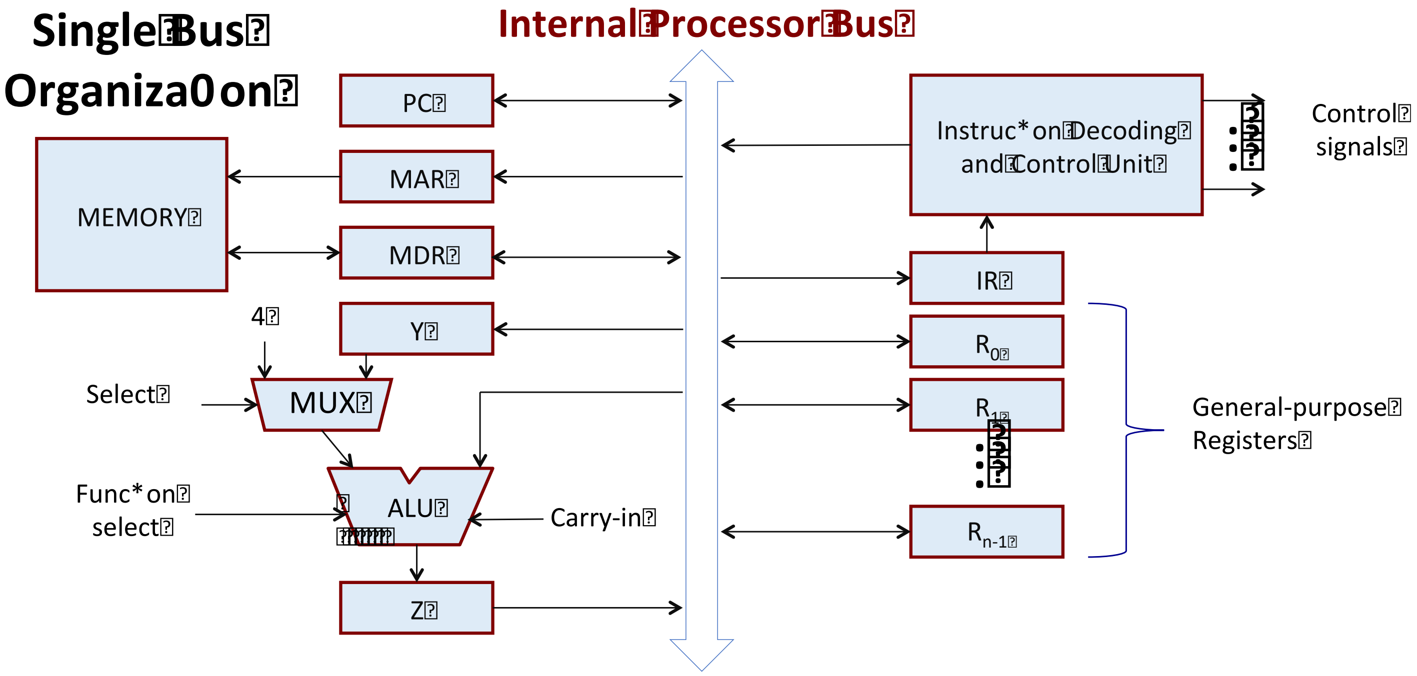
12. HALT

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

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

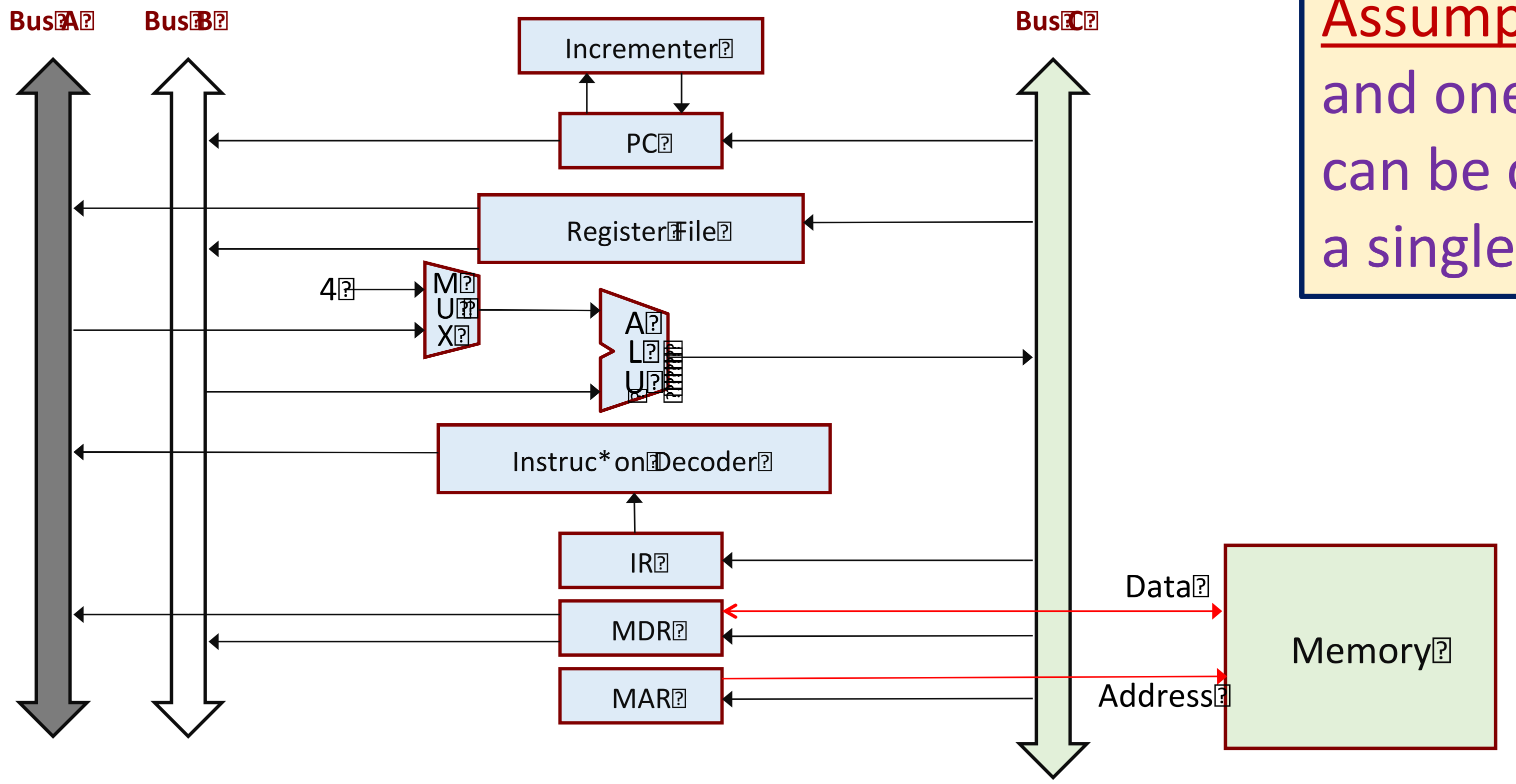
T3: MDR_{out}, IR_{in}

T4: END



Practice Assignment 1

- Write down the micro-operations for the 12 instructions with respect to the 3-bus architecture.



Assumption: Two register read and one register write operations can be carried out concurrently in a single time step.

Practice Assignment 2

- Complete the following table for both the 1-bus and 3-bus architectures.
  - Ins-1 to Ins-12 indicates the instructions.
  - The entries in the table will contain the corresponding micro-operations.

	Ins-1	Ins-2	Ins-3	Ins-4	Ins-5	Ins-6	Ins-7	Ins-8	Ins-9	Ins-10	Ins-11	nIs-12
T1												
T2												
T3												
T4												
T5												
T6												
T7												

Hardwired and Microprogrammed Control Unit Design

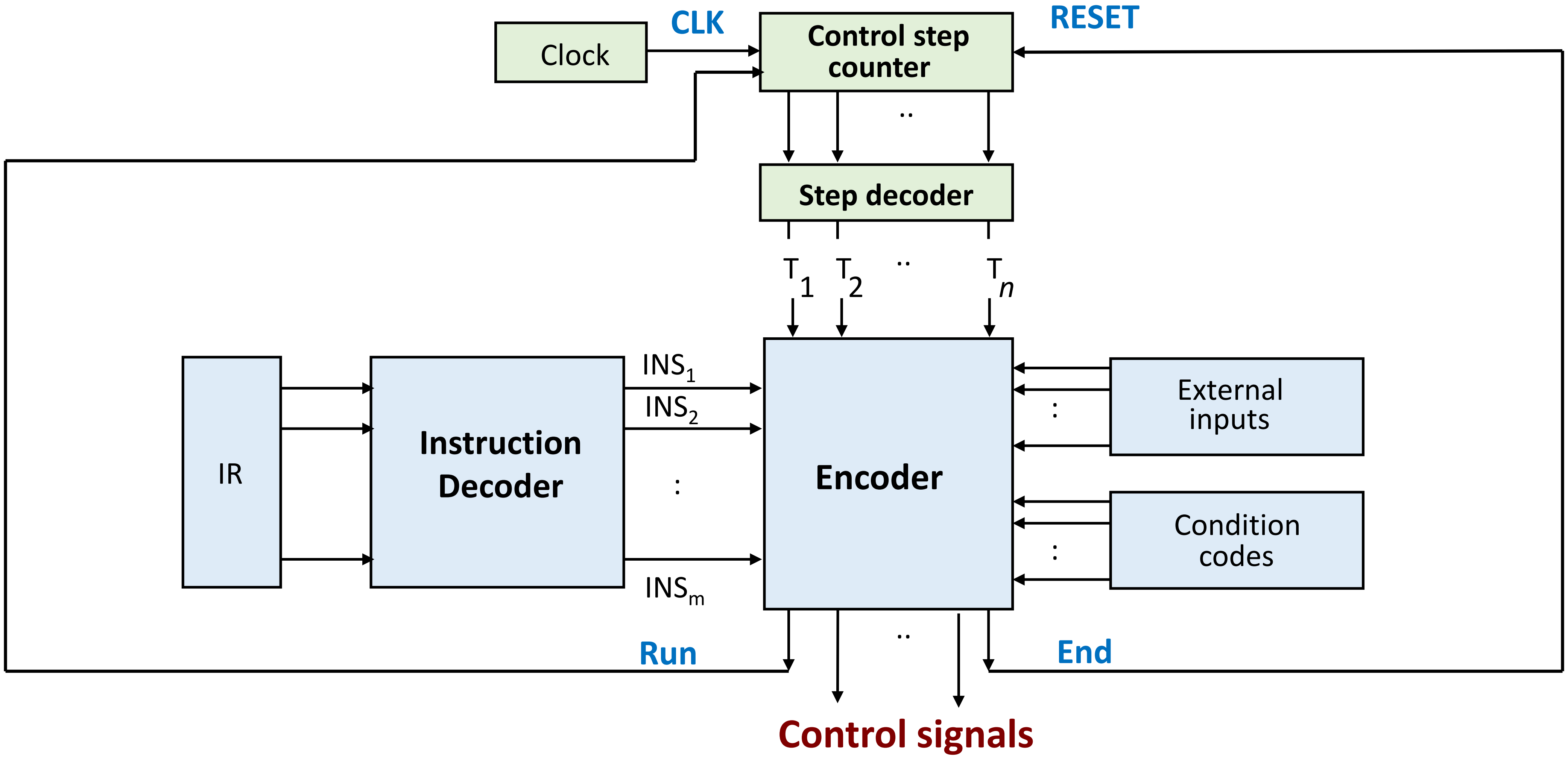
Introduction

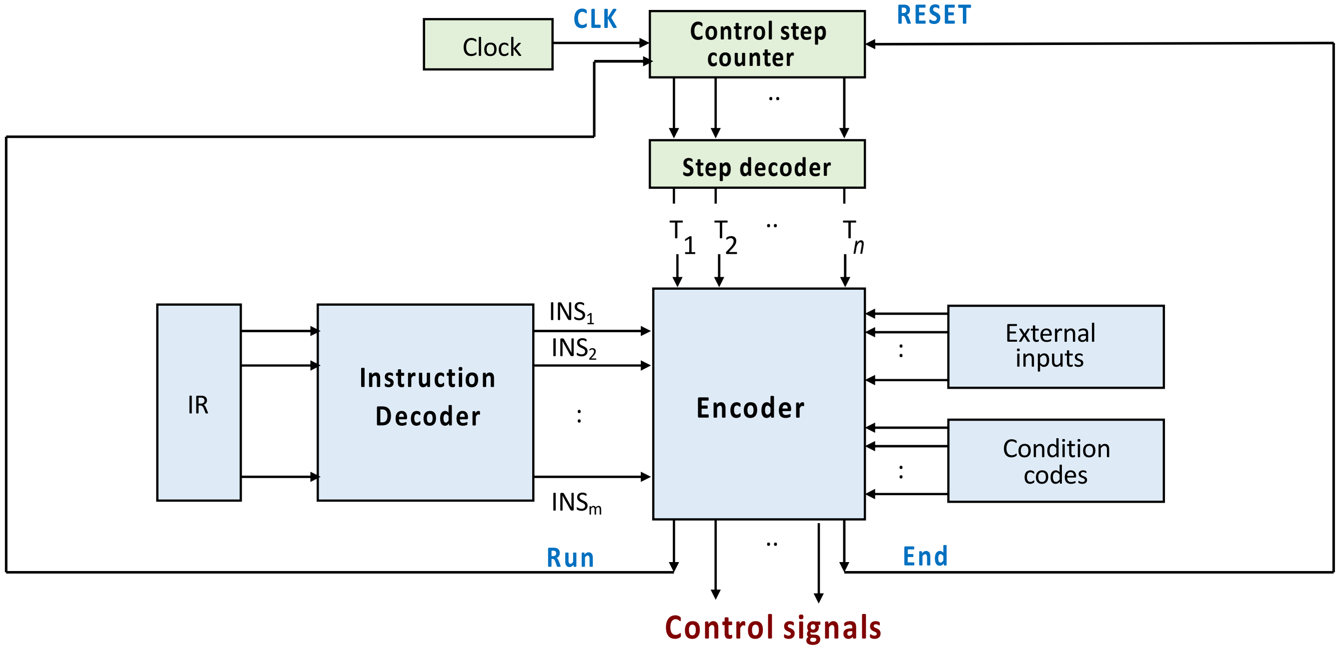
- To execute an instruction, the processor must generate control signals for the data path in proper sequence.
  - Example: ADD R1, R2
    a) $R1_{out}$, Y_{in}, SelectY
    b) $R2_{out}$, ADD, Z_{in}
    c) Z_{out}, $R1_{in}$
- Two alternate approaches:
  1. Hardwired control unit design
  2. Microprogrammed control unit design

Hardwired Control Unit Design

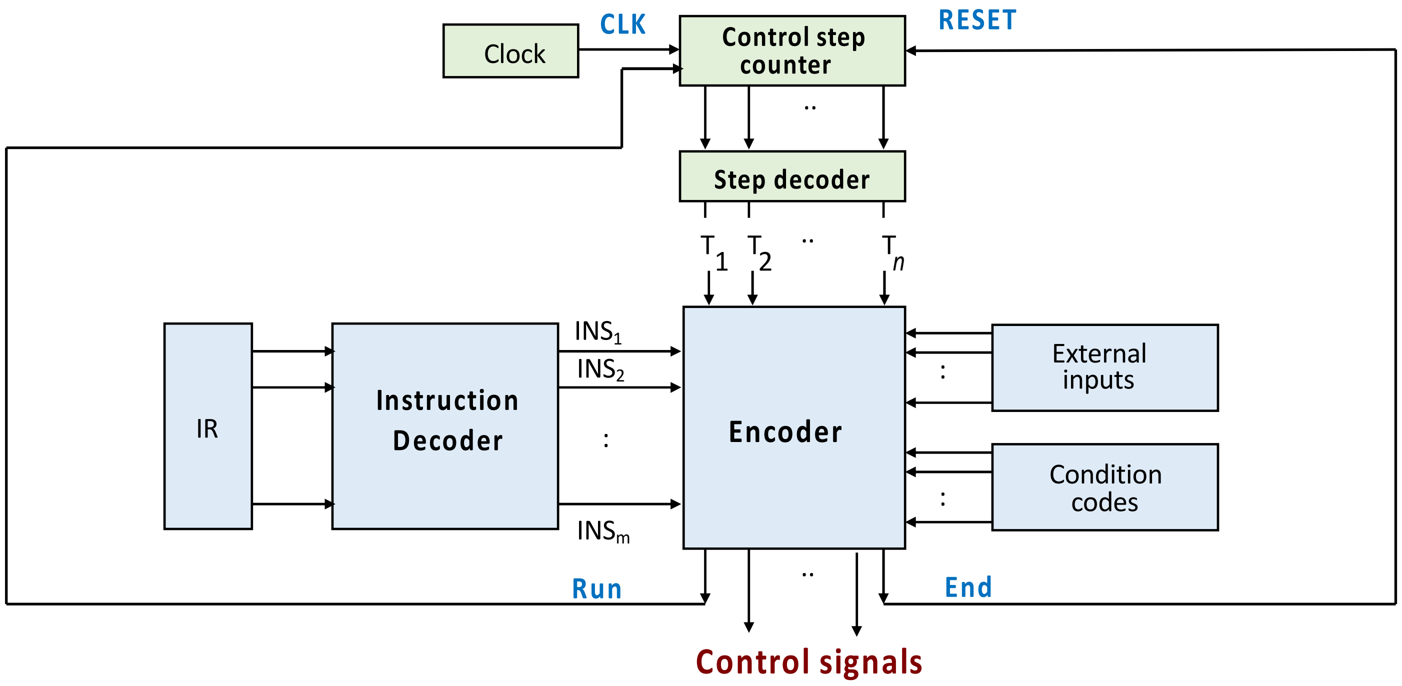
- What does the control unit do?
  - It generates the control signals in proper sequence so as to execute any given instruction.
  - We have already seen some example control signals for some instructions.
- Basic question?
  - For an instruction I_k and time step T_m, what are the control signals that need to be generated?
- What is hardwired control unit?
  - The control signals are generated by specific hardware circuits that cannot be easily modified.
  - It is fixed or hardwired for a given <architecture, ISA> pair.

Overall Schematic Diagram

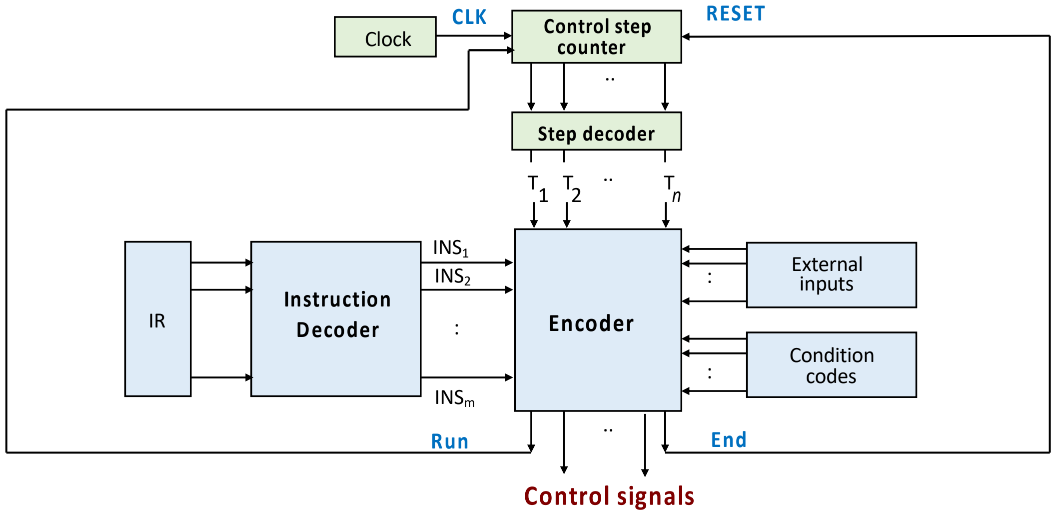




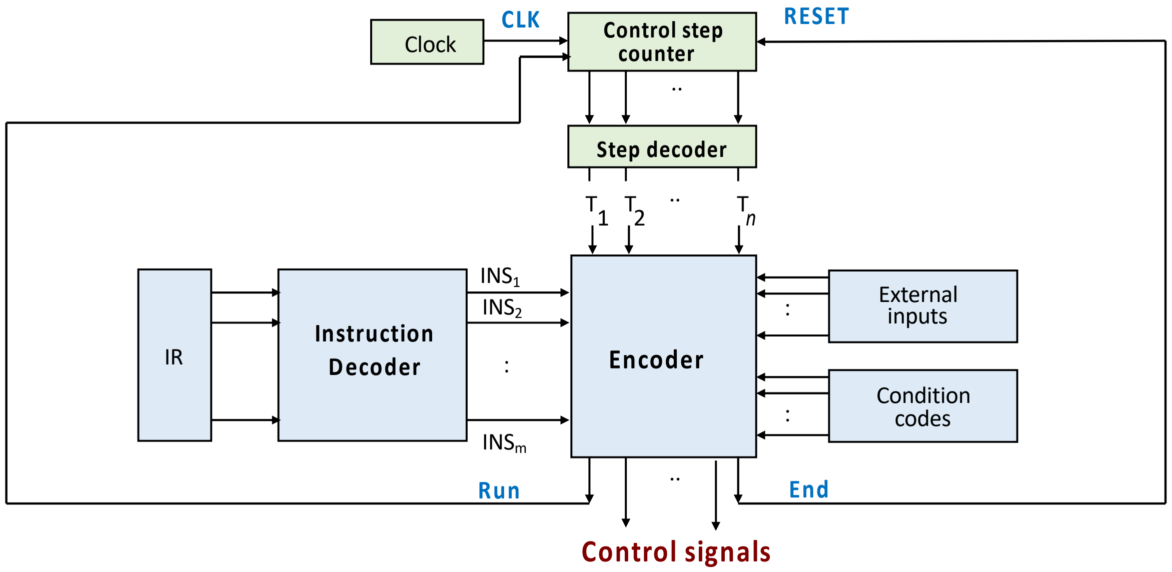
- Basic Principle of Operation:
  - The *step decoder* provides separate signal for each step (T_1, T_2, T_3, etc.)..
  - The *instruction decoder* has separate output lines for each machine instruction.
  - For any instruction loaded in IR, one of the output line INS_1, …, INS_m is set to 1 (others are 0).
  - The input signal to the *encoder* block are combined to generate individual control signals.
  - The signal *RUN* causes the counter to be incremented by 1, stops counting when *RUN = 0*.
- Assumption:
  - Each time step is completed in one clock cycle.
  - The control step counter keeps track of the time step.



- The control signals are determined by the following information:
  - Content of control step counter
  - Content of instruction register
  - Content of conditional code flags
  - External input signals such as WMFC



- The *Encoder* circuit is a combinational circuit which generates control signals depending on the inputs provided.
- The *Step Decoder* generates separate signal lines 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.
- The *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 (Hardwired Control)

ADD 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}

T5: $R2_{out}$, SelectY, Add, Z_{in}

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

ADD R1,LOCA

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

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

T3: MDR_{out}, IR_{in}

T4: Address field of IR_{out}, MAR_{in}, Read

T5: $R1_{out}$, Y_{in}, WMFC

T6: MDR_{out}, SelectY, Add, Z_{in}

T7: Z_{out}, $R1_{in}$, End

LOAD R1,LOCA

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

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

T3: MDR_{out}, IR_{in}

T4: Address field of IR_{out}, MAR_{in}, Read

T5: WMFC

T6: MDR_{out}, $R1_{in}$, End

BR Label

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

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

T3: MDR_{out}, IR_{in}

T4: Offset-field-of-IR_{out}, SelectY, Add, Z_{in}

T5: Z_{out}, PC_{in}, End

INC R1

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

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

T3: MDR_{out}, IR_{in}

T4: $R1_{out}$, Select4, Add, Z_{in}

T5: Z_{out}, $R1_{in}$, End

	ADD R1,R2	ADD R1,LOCA	LOAD R1,LOCA	BR Label	INC R1
	I1	I2	I3	I4	I5
T1	PC_{out}, MAR_{in}, Read, Select4, ADD, Z_{in}	PC_{out}, MAR_{in}, Read, Select4, ADD, Z_{in}	PC_{out}, MAR_{in}, Read, Select4, ADD, Z_{in}	PC_{out}, MAR_{in}, Read, Select4, ADD, Z_{in}	PC_{out}, MAR_{in}, Read, Select4, ADD, Z_{in}
T2	Z_{out}, PC_{in}, Y_{in}, WMFC	Z_{out}, PC_{in}, Y_{in}, WMFC	Z_{out}, PC_{in}, Y_{in}, WMFC	Z_{out}, PC_{in}, Y_{in}, WMFC	Z_{out}, PC_{in}, Y_{in}, WMFC
T3	MDR_{out}, IR_{in}	MDR_{out}, IR_{in}	MDR_{out}, IR_{in}	MDR_{out}, IR_{in}	MDR_{out}, IR_{in}
T4	$R1_{out}$, Y_{in}	Address field of IR_{out}, MAR_{in}, Read	Address field of IR_{out}, MAR_{in}, Read	Offset field of IR_{out}, SelectY, ADD, Z_{in}	$R1_{out}$, Select4, ADD, Z_{in}
T5	$R2_{out}$, SelectY, ADD, Z_{in}	$R1_{out}$, Y_{in}, WMFC	WMFC	Z_{out}, PC_{in}, END	Z_{out}, $R1_{in}$, END
T6	Z_{out}, $R1_{in}$, END	MDR_{out}, SelectY, ADD, Z_{in}	MDR_{out}, $R1_{in}$, END		
T7		Z_{out}, $R1_{in}$, END			

Logic for Generation of Some of the Control Signals

PC_{in} = T2 + T5.I4

PC_{out} = T1

MAR_{in} = T1 + T4.(I2 + I3)

Z_{in} = T1 + T4.(I4 + I5) + T6.I2

Z_{out} = T2 + T5.(I4 + I5) + T6.I1 + T7.I2

MDR_{out} = T3 + T6.(I2 + I3)

IR_{in} = T3

Y_{in} = T2 + T4.I1 + T5.I2

Select4 = T1 + T4.I5

SelectY = T4.I4 + T5.I1 + T6.I2

END = T5.(I4 + I5) + T6.(I1 + I3) + T7.I2

Rx_{in} = T5.I5 + T6.(I1 + I3) + T7.I2

Rx_{out} = T4.(I1 + I5) + T5.(I1 + I2)

WMFC = T2 + T5.(I2 + I3)

Design of the Encoder Block

- Implement the logic functions to generate the control signals.
    - Using logic gates or any other hardware module.

- This method of generating control signals is very fast.

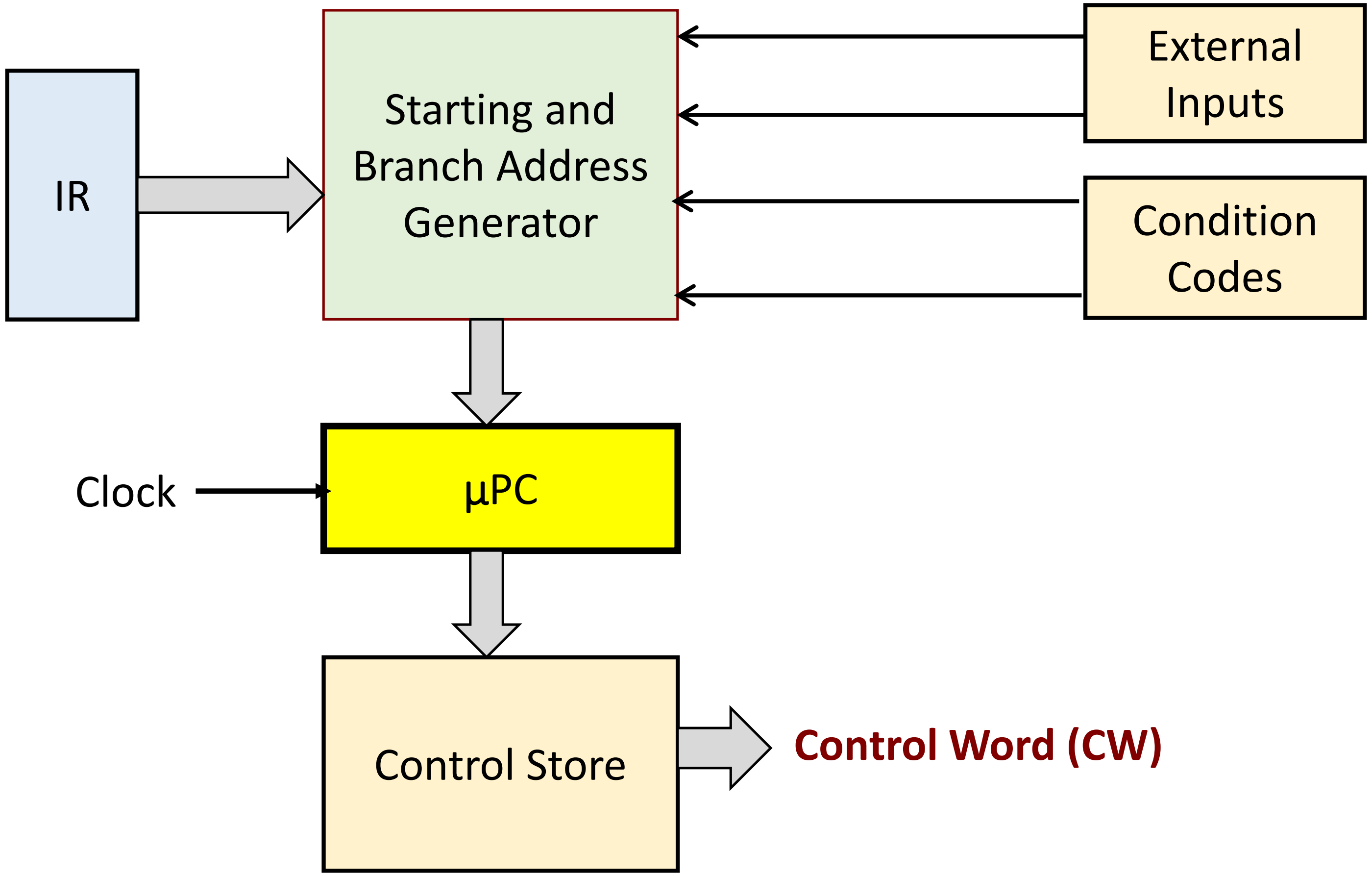
Practice Assignment 3

- Design the hardwired control unit for the 12 instructions as given for:
    a) 1-bus architecture
    b) 3-bus architecture

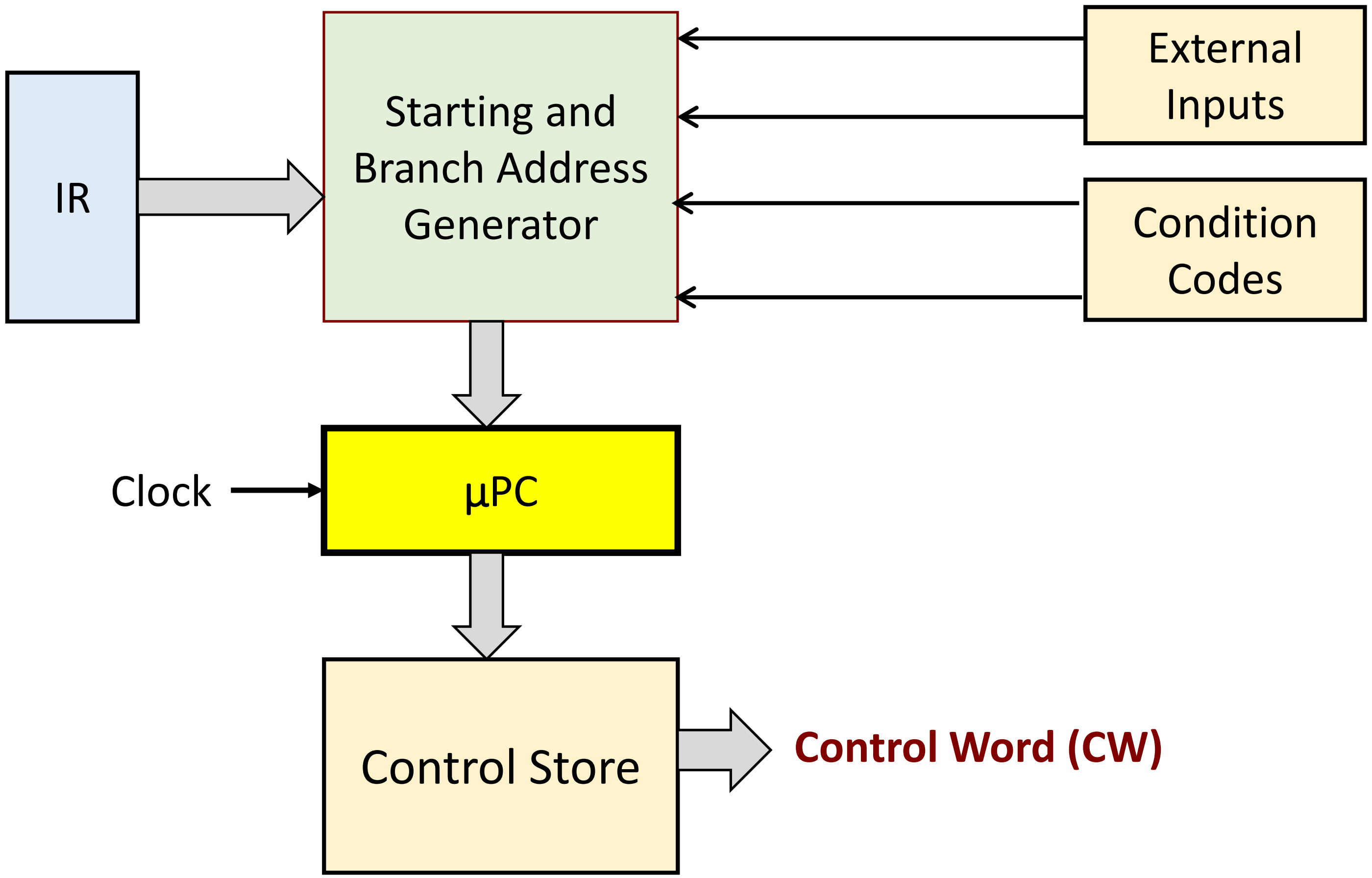
Generation of Control Signals (Microprogrammed Control)

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*.


IR
Starting and Branch Address Generator
External Inputs
Condition Codes
Clock
μPC
Control Store
Control Word (CW)

- Control-unit generates the control signals for an instruction by sequentially reading CWs of corresponding microroutine from CS.
- The *micro-program counter* (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 *CS*.



Control Store Contents for "ADD R1, R2"

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

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}

T5: $R2_{out}$, SelectY, Add, Z_{in}

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

Control Store Contents for “BR Label”

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

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

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

T3: MDR_{out}, IR_{in}

T4: Offset-field-of-IR_{out}, SelectY, Add, Z_{in}

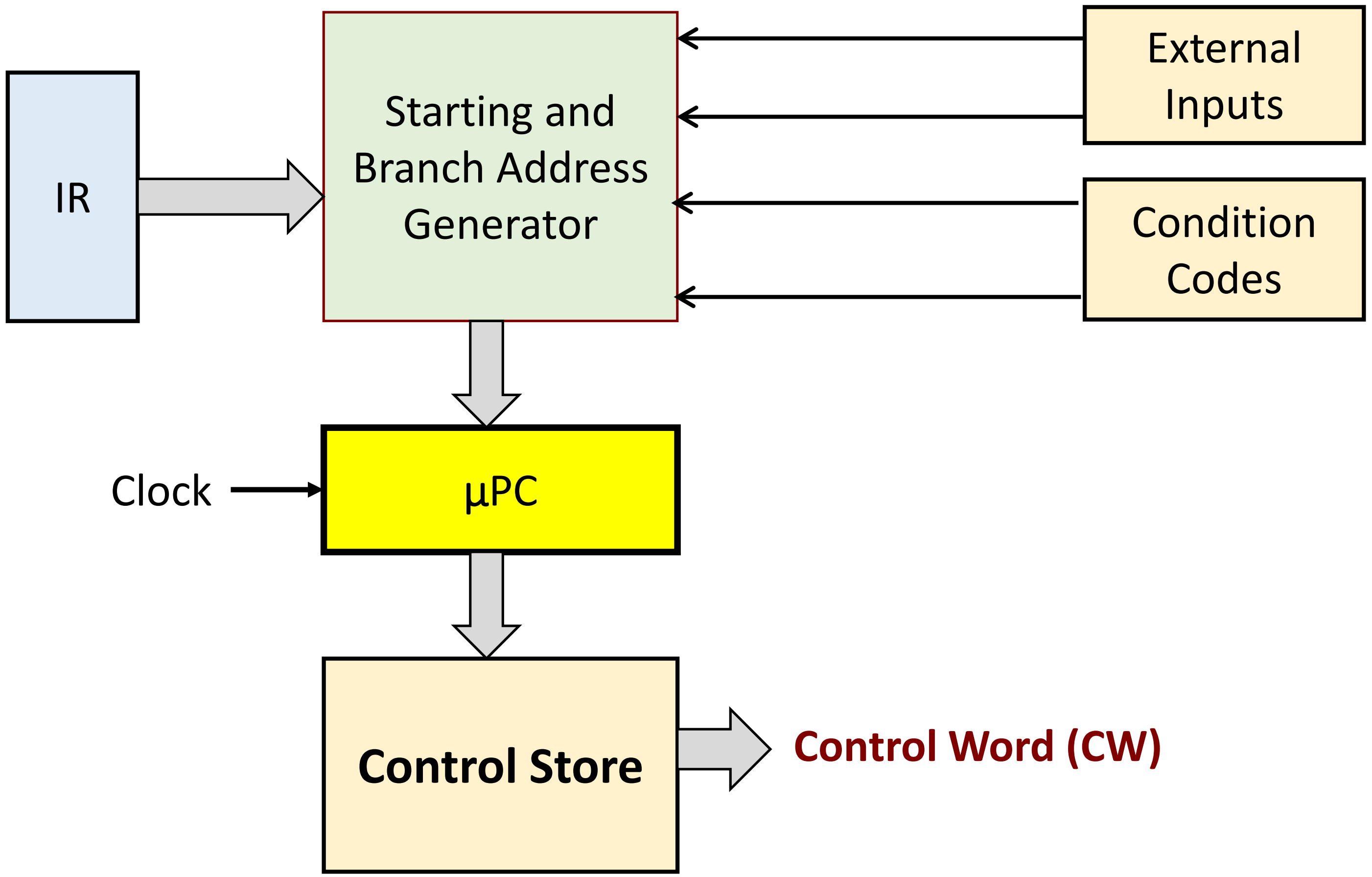
T5: Z_{out}, PC_{in}, End

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*.
  - An example is shown on the next slide.

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


IR
Starting and Branch Address Generator
External Inputs
Condition Codes
Clock
μPC
Control Store
Control Word (CW)

Microprogram Sequencing

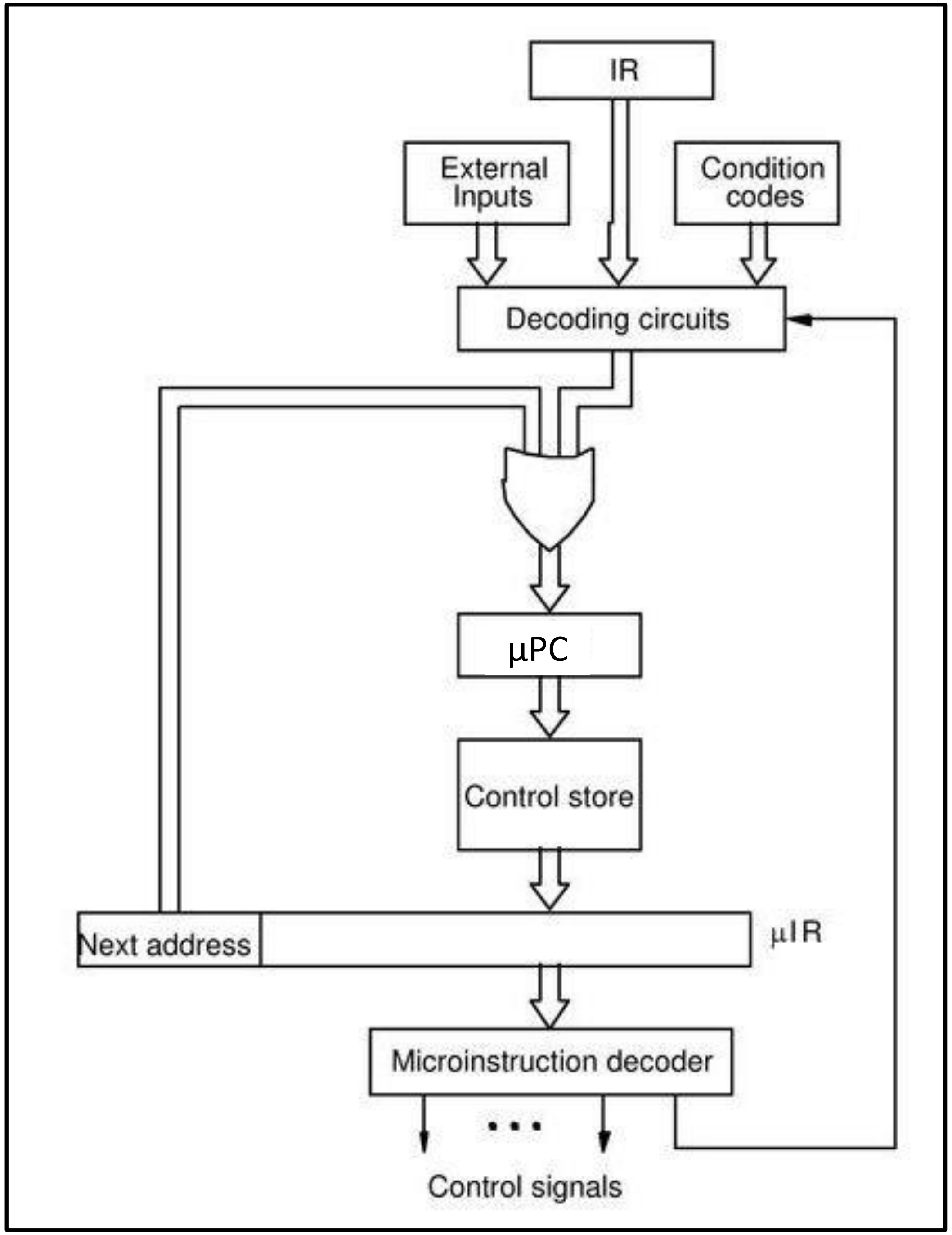
- If all microprograms require only straightforward sequential execution of microinstructions except for branches, letting a μPC govern the sequencing would be efficient.

- 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; few limitations in assigning addresses to microinstructions.
- Cons: additional bits required for the address field.



From the instruction opcode, some constant bits are appended (both at the left and right) to get the starting address of the corresponding micro-routine in control store.

Illustrative Example

- Some example instructions and their 4-bit opcodes:

	Opcodes	*Starting Address of Micro-routine*
ADD	0010	0 0010 000
SUB	0011	0 0011 000
LOAD	0100	0 0100 000
STORE	0101	0 0101 000
BR	1010	0 1010 000

Practice Assignment 4

Design the microprogrammed control unit for the 12 instructions as given for the 1-bus architecture.

In particular, show the micro-routines for the various instructions, and the detailed contents of CS.

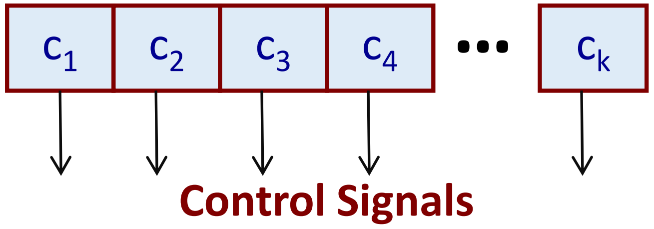
Horizontal versus Vertical Microinstruction Encoding

- 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.

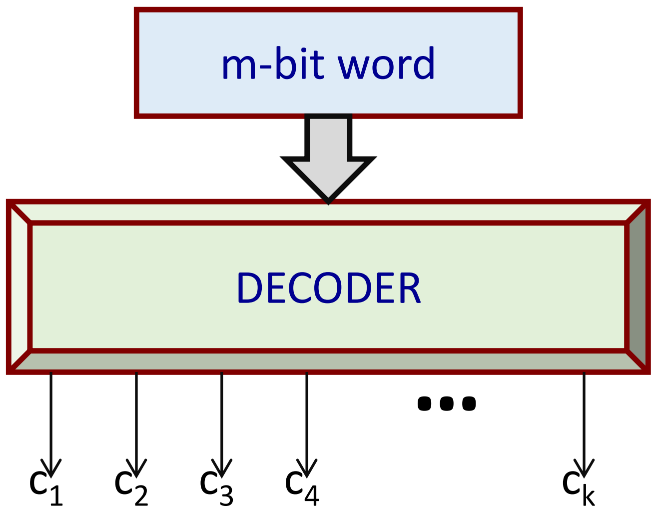
0	1	0	1	1	0

→ 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.

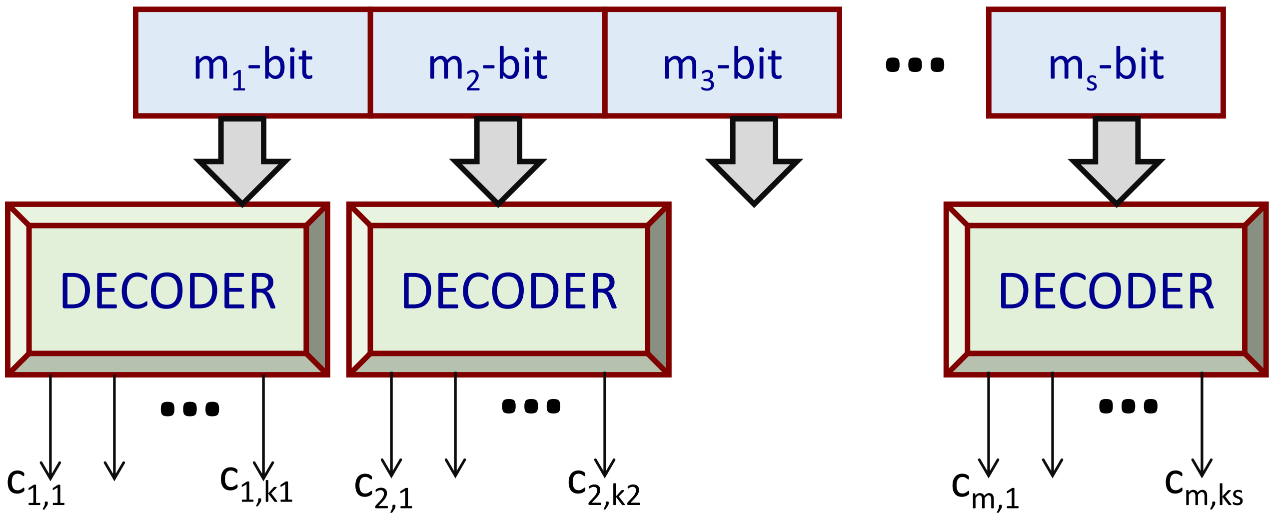
(b) Vertical Micro-instruction Encoding

- 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 $k \leq 2^m$.
- 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, …, 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 1

- 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.

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