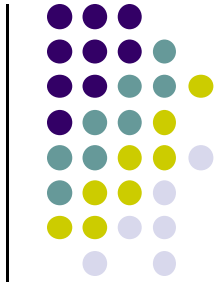
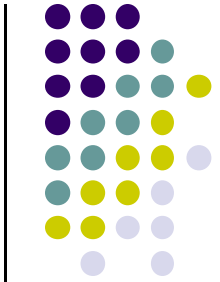


Unit III - Basic Processing Unit

C.Bala Subramanian,
AP/CSE,
KLU

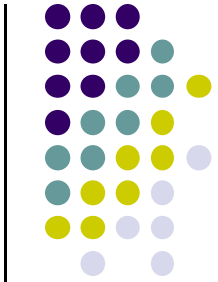
Overview



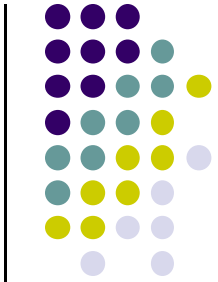
CISC
RISC

- Instruction Set Processor (ISP)
- Central Processing Unit (CPU)
- A typical computing task consists of a series of steps specified by a sequence of machine instructions that constitute a program.
- An instruction is executed by carrying out a sequence of more rudimentary operations.

Some Fundamental Concepts



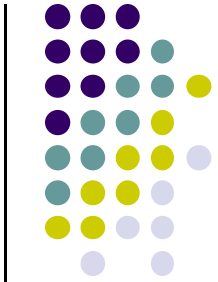
7.1 Fundamental Concepts



- Processor fetches one instruction at a time and perform the operation specified.

- Instructions are fetched from successive memory locations until a branch or a jump instruction is encountered.
- Processor keeps track of the address of the memory location containing the next instruction to be fetched using Program Counter (PC).
- Instruction Register (IR)

Executing an Instruction



- Fetch the contents of the memory location pointed to by the PC. The contents of this location are loaded into the IR (fetch phase).

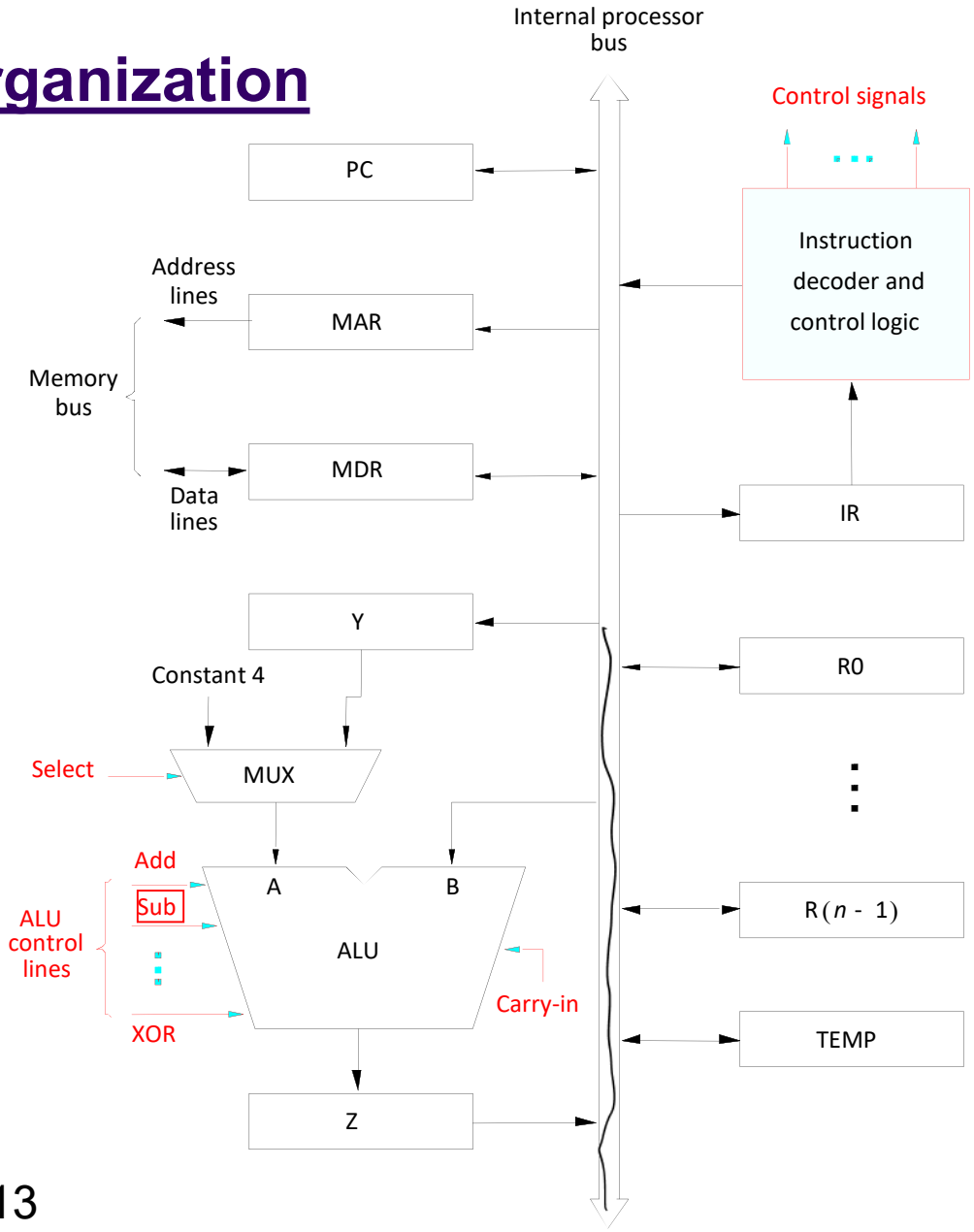
$$IR \leftarrow [[PC]]$$

- Assuming that the memory is byte addressable, increment the contents of the PC by 4 (fetch phase).

$$PC \leftarrow [PC] + 4$$

- Carry out the actions specified by the instruction in the IR (execution phase).

Processor Organization

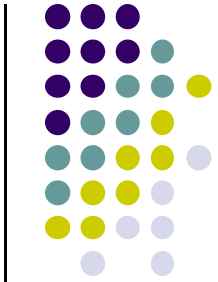


Datapath

Textbook Page 413

Figure 7.1. Single-bus organization of the datapath inside a processor.

Executing an Instruction



- Transfer a word of data from one processor register to another or to the ALU.
- Perform an arithmetic or a logic operation and store the result in a processor register.
- Fetch the contents of a given memory location and load them into a processor register.
- Store a word of data from a processor register into a given memory location.

7.1.1 Register Transfers

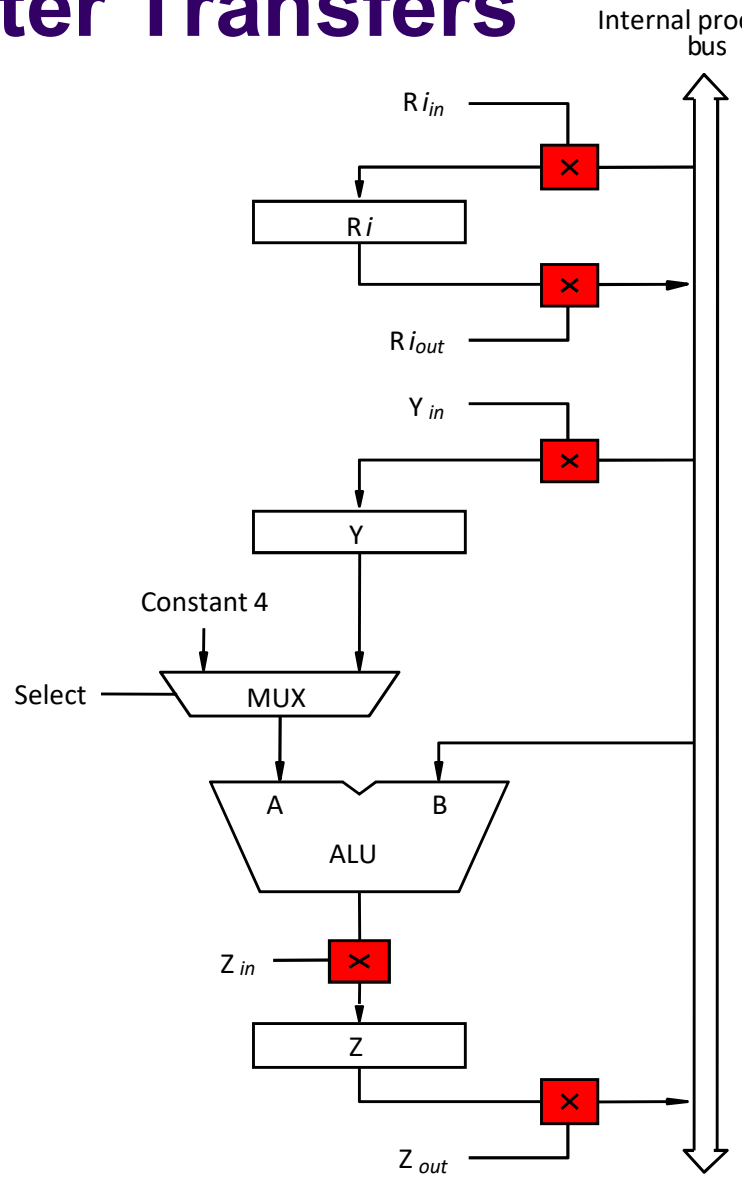


Figure 7.2. Input and output gating for the registers in Figure 7.1.

Register Transfers

- All operations and data transfers are controlled by the processor clock.

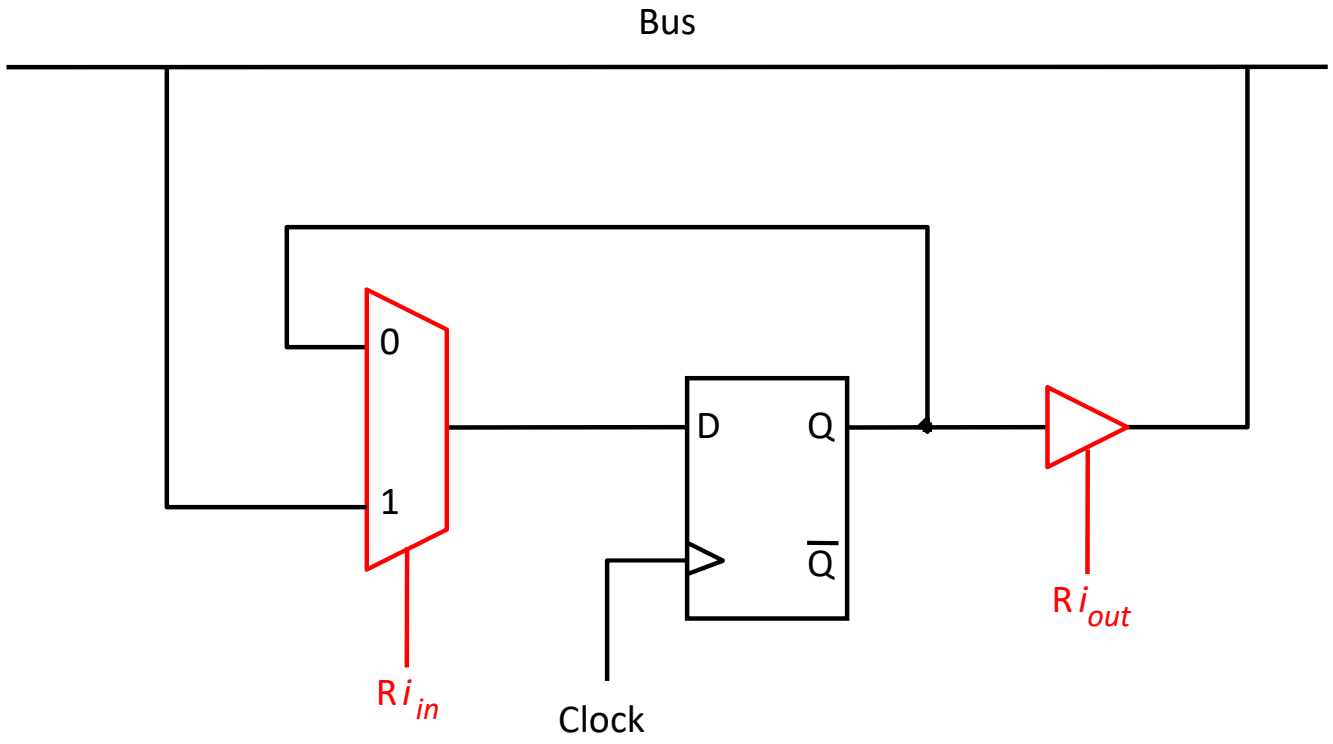
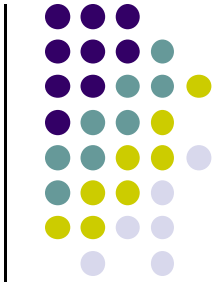


Figure 7.3. Input and output gating for one register bit.

7.1.2 Performing an Arithmetic or 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.
- What is the sequence of operations to add the contents of register R1 to those of R2 and store the result in R3?

1. R1out, Yin
2. R2out, SelectY, Add, Zin
3. Zout, R3in

7.1.3 Fetching a Word from Memory

- Address into MAR; issue Read operation; data into MDR.

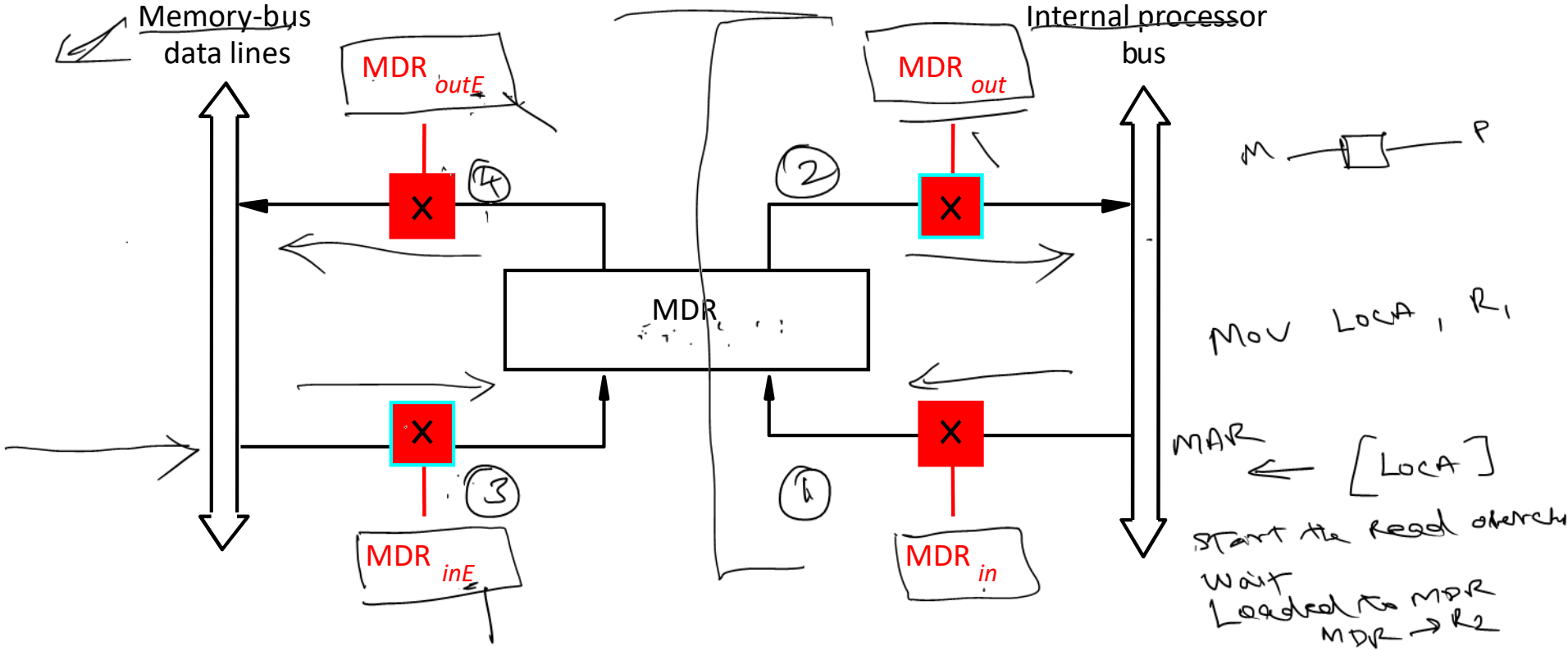
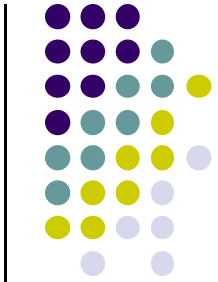


Figure 7.4. Connection and control signals for register MDR.

Fetching a Word from Memory



- The response time of each memory access varies (cache miss, memory-mapped I/O,…).
- To accommodate this, the processor waits until it receives an indication that the requested operation has been completed (Memory-Function-Completed, MFC).
- Move (R1), R2
  - MAR ← [R1]
  - Start a Read operation on the memory bus
  - Wait for the MFC response from the memory
  - Load MDR from the memory bus
  - R2 ← [MDR]

Move (R1), R2

$R1_{out}$, MAR_{in}, Read

MDR_{inE}, WFMC

MDR_{out}, $R2_{in}$

Timing

Assume MAR is always available on the address lines of the memory bus.

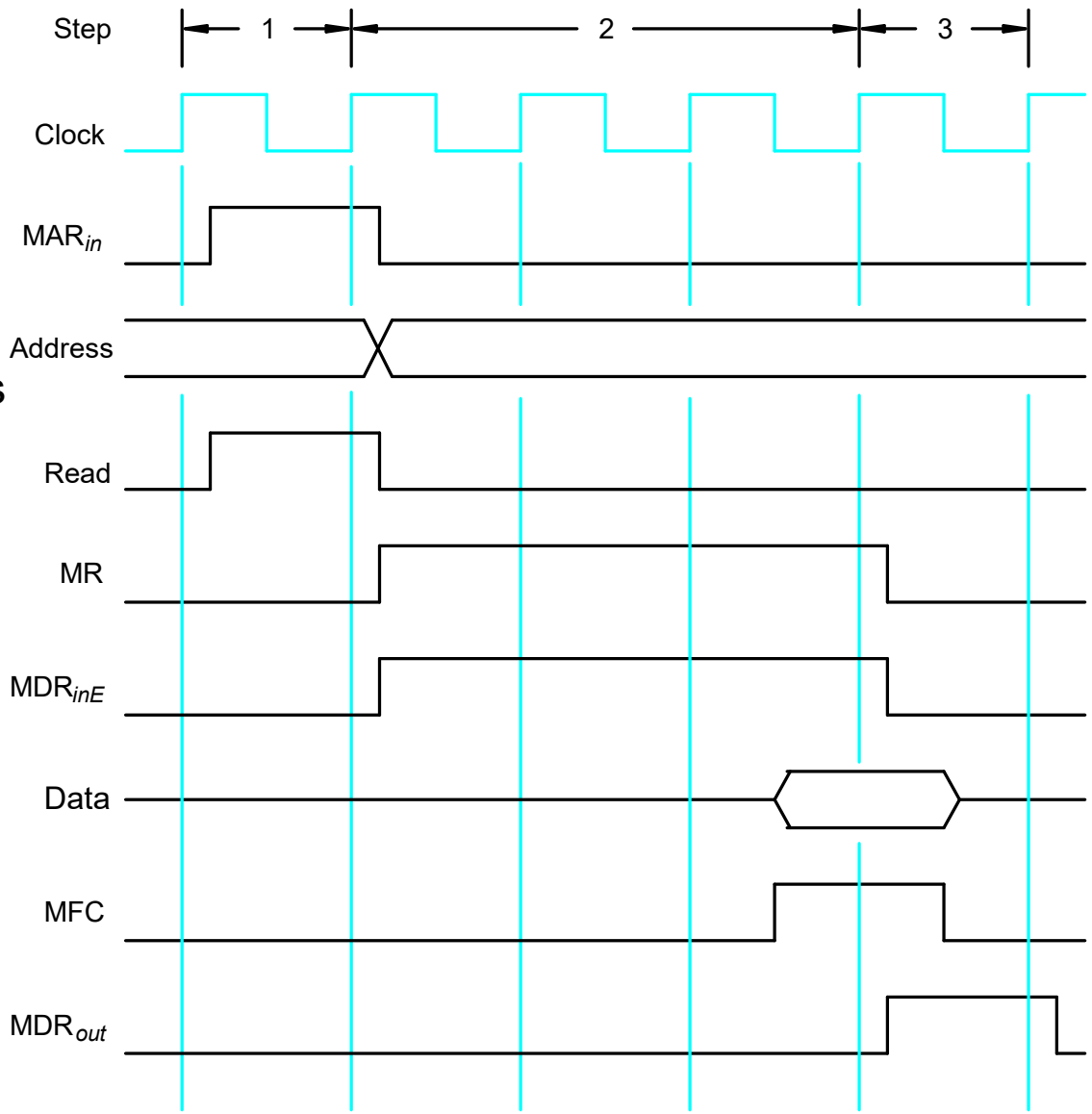


Figure 7.5. Timing of a memory Read operation.

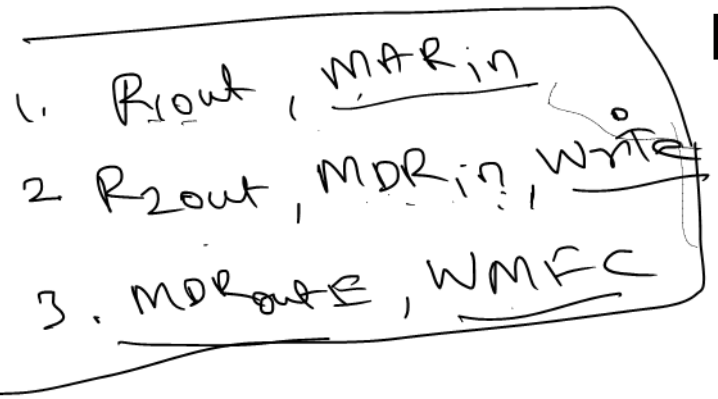
7.1.4 Storing a Word in Memory

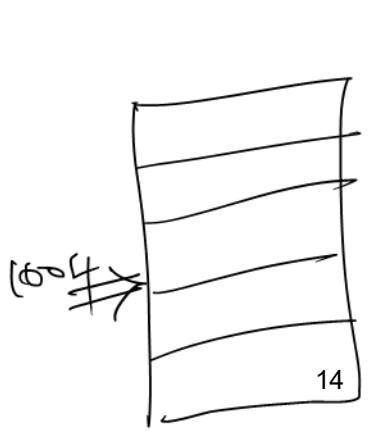
Move R2, (R1)

$R1_{out}$, MAR_{in}

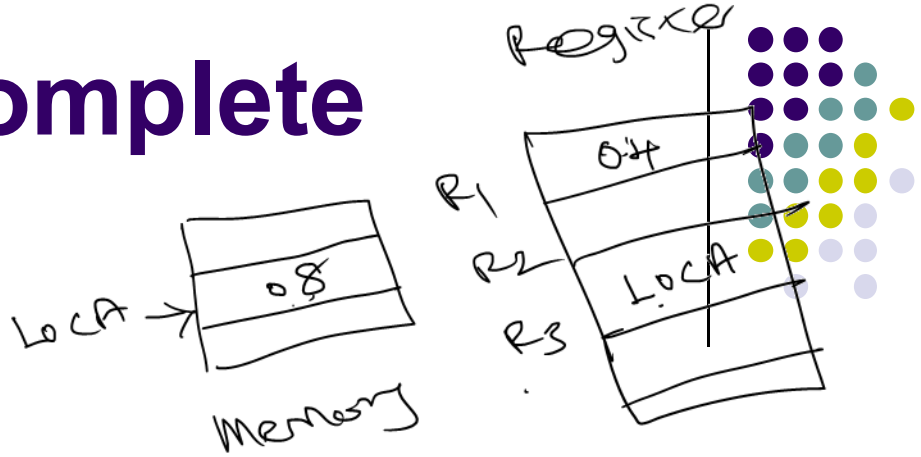
$R2_{Out}$, MDR_{in}, Write

MDR_{outE}, WMFC





Execution of a Complete Instruction



- Add (R3), R1
- Fetch the instruction
- Fetch the first operand (the contents of the memory location pointed to by R3)
- Perform the addition
- Load the result into R1

Architecture

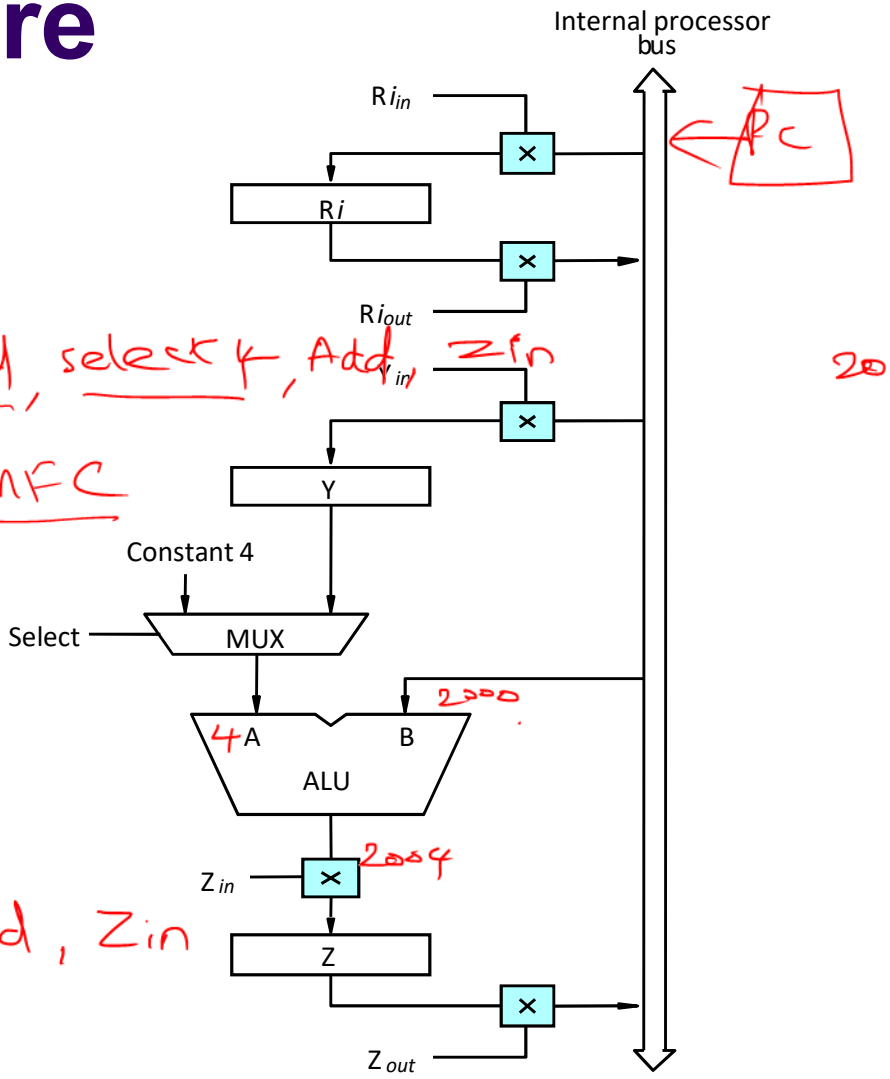


Figure 7.2. Input and output gating for the registers in Figure 7.1.

Execution of a Complete Instruction

Add (R3), R1

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

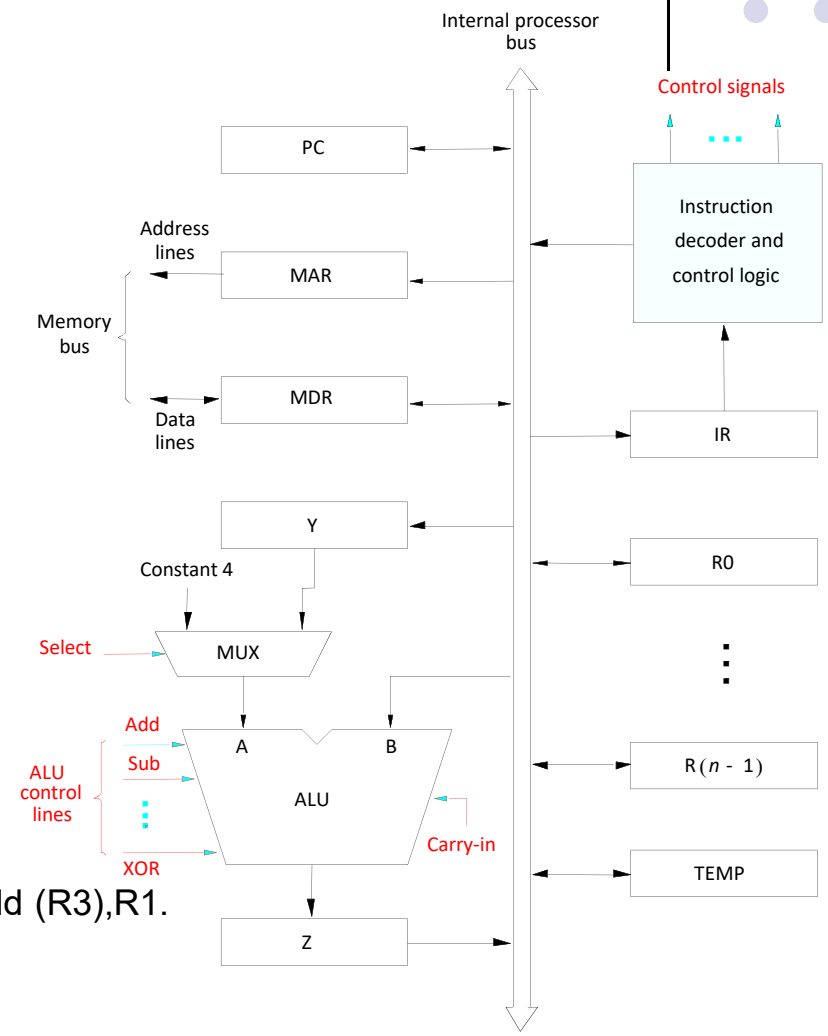
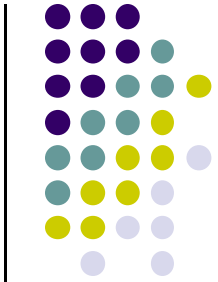


Figure 7.6. Control sequence for execution of the instruction Add (R3),R1.

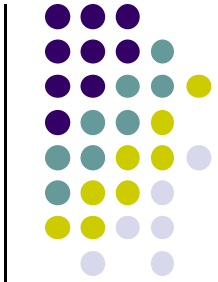
Figure 7.1. Single-bus organization of the datapath inside a processor.

Execution of Branch Instructions



- A branch instruction replaces the contents of PC with the branch target address, which is usually obtained by adding an offset X given in the branch instruction.
- The offset X is usually the difference between the branch target address and the address immediately following the branch instruction.
- Conditional branch

Execution of Branch Instructions



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

Figure 7.7. Control sequence for an unconditional branch instruction.

Multiple-Bus Organization

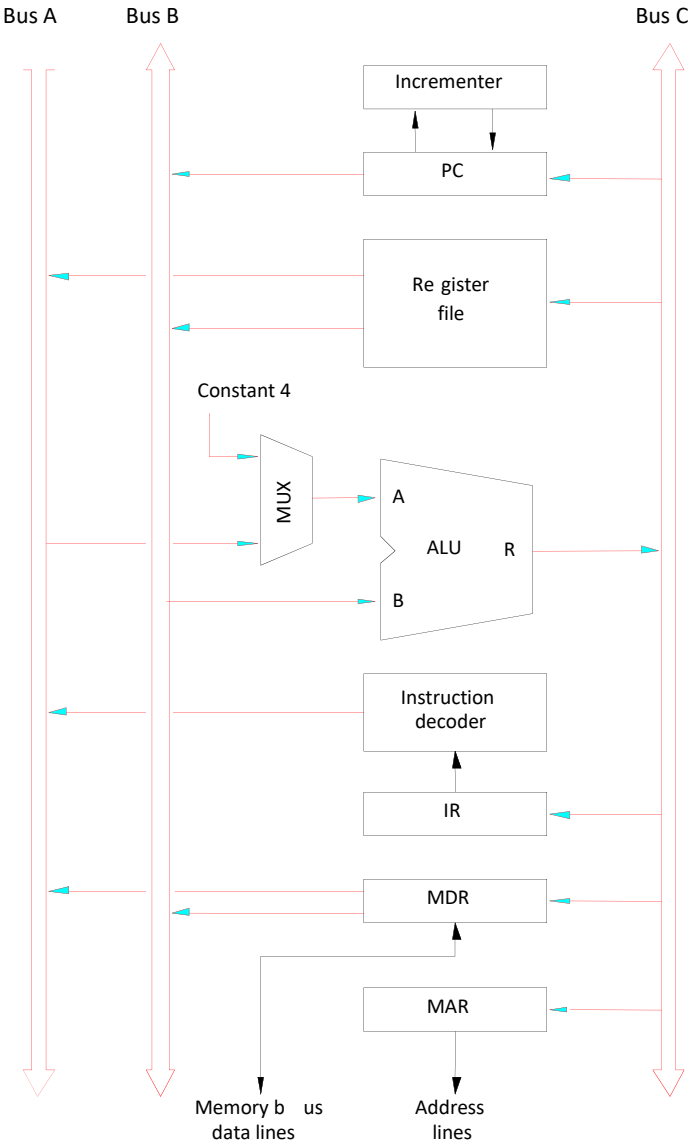
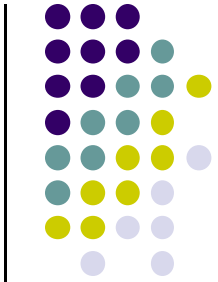


Figure 7.8. Three-b us or g anization of the datapath.

Multiple-Bus Organization



- Add R4, R5, R6

Step	Action
1	PC_{out}, R=B, MAR_{in}, Read, IncPC
2	WMFC
3	MDR_{outB}, R=B, IR_{in}
4	$R4_{outA}$, $R5_{outB}$, SelectA, Add, $R6_{in}$, End

Figure 7.9. Control sequence for the instruction. Add R4,R5,R6, for the three-bus organization in Figure 7.8.

Quiz

- What is the control sequence for execution of the instruction

  Add R1, R2

  including the instruction fetch phase? (Assume single bus architecture)

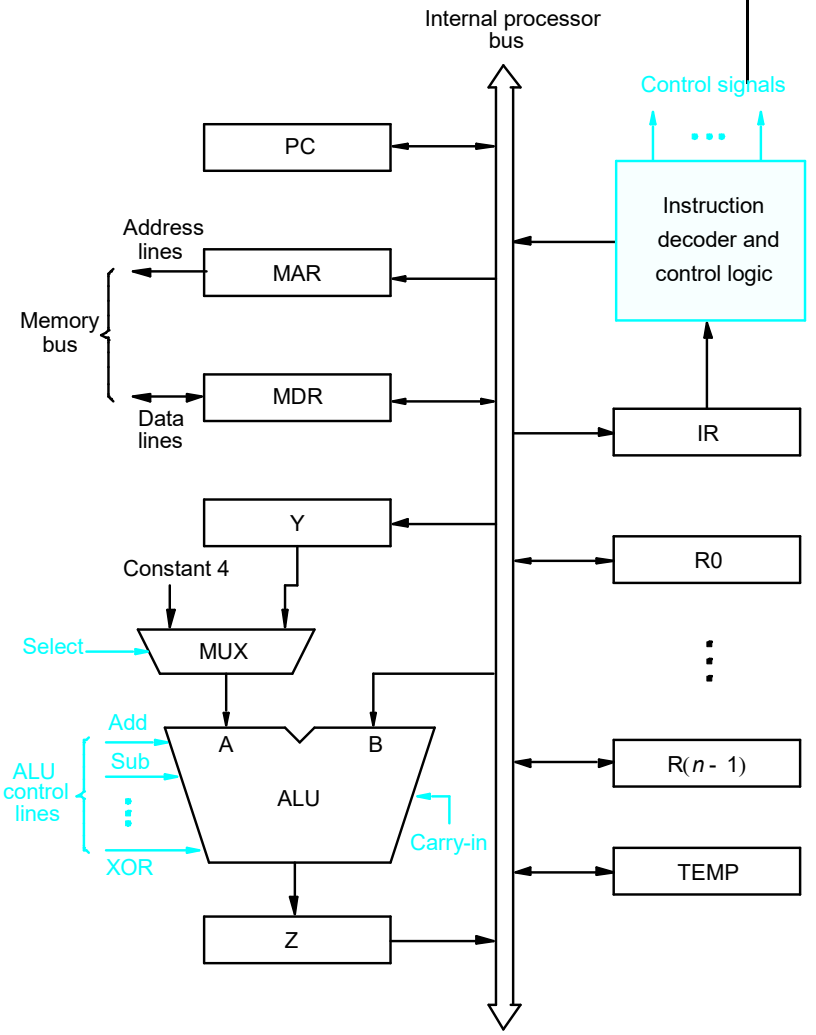
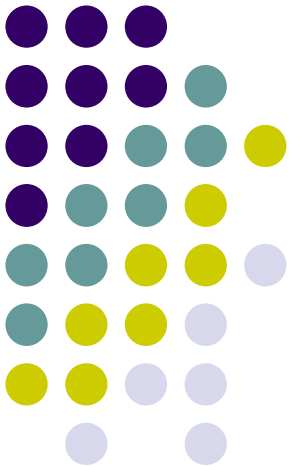
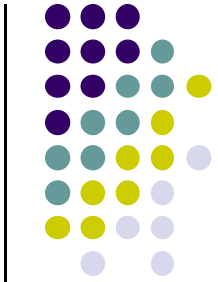


Figure 7.1. Single-bus organization of the datapath inside a proc

Hardwired Control

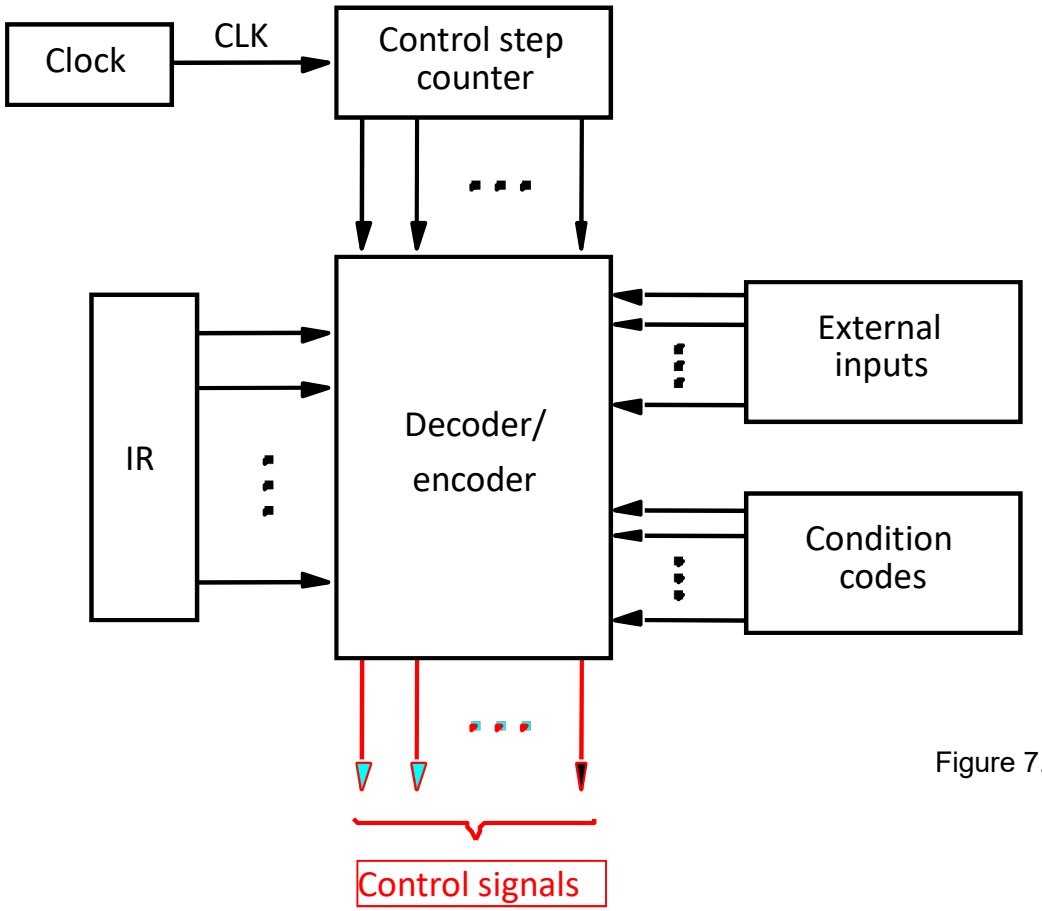


Overview



- To execute instructions, the processor must have some means of generating the control signals needed in the proper sequence.
- *Two categories*: hardwired control and microprogrammed control
- Hardwired system can operate at high speed; but with little flexibility.

Control Unit Organization



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	$R3_{out}$, MAR_{in} , Read
5	$R1_{out}$, Y_{in} , WMFC
6	MDR_{out} , SelectY, Add, Z_{in}
7	Z_{out} , $R1_{in}$, End

Figure 7.6. Control sequence for execution of the instruction Add (R3),R1.

Figure 7.10. Control unit organization.

Detailed Block Description

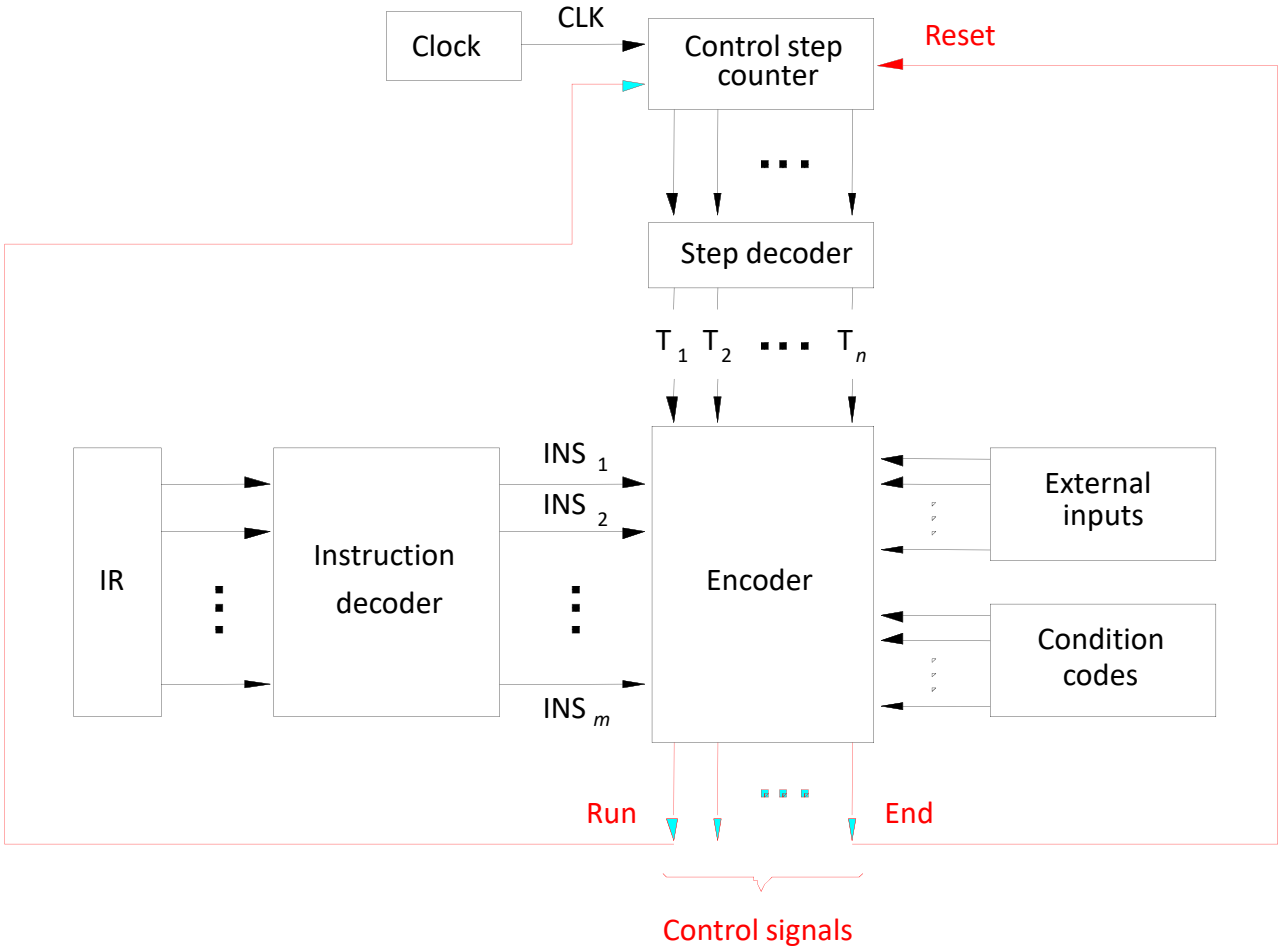


Figure 7.11. Separation of the decoding and encoding functions.

Generating Z_{in}

- $Z_{in} = T_1 + T_6 \bullet ADD + T_4 \bullet BR + \ldots$

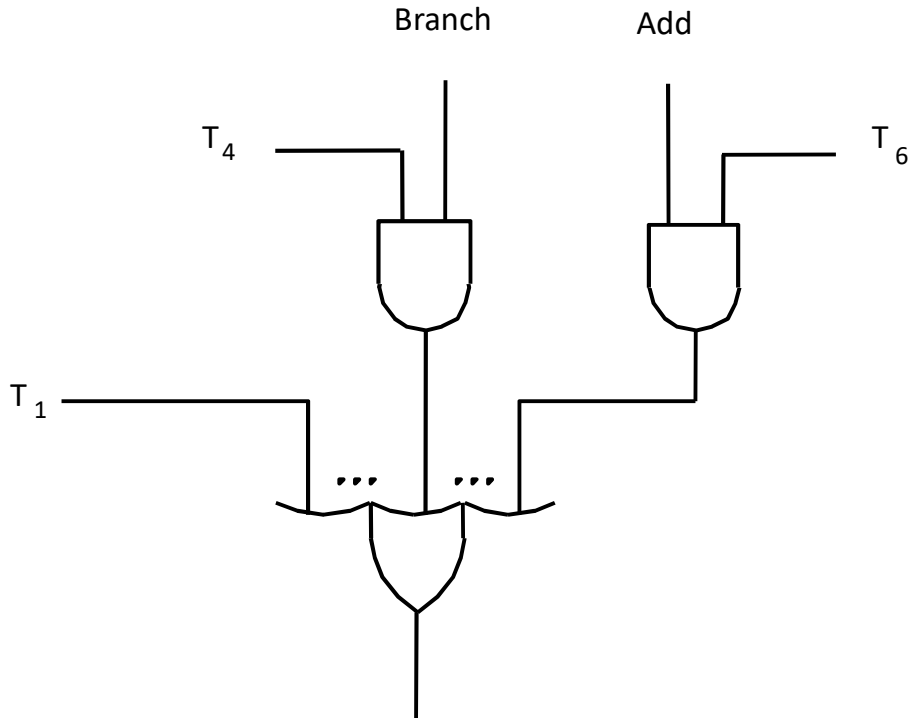


Figure 7.12. Generation of the Z_{in} control signal for the processor in Figure 7.1.

Generating End

- End = $T_7 \bullet ADD + T_5 \bullet BR + (T_5 \bullet N + T_4 \bullet \overline{N}) \bullet BRN + \ldots$

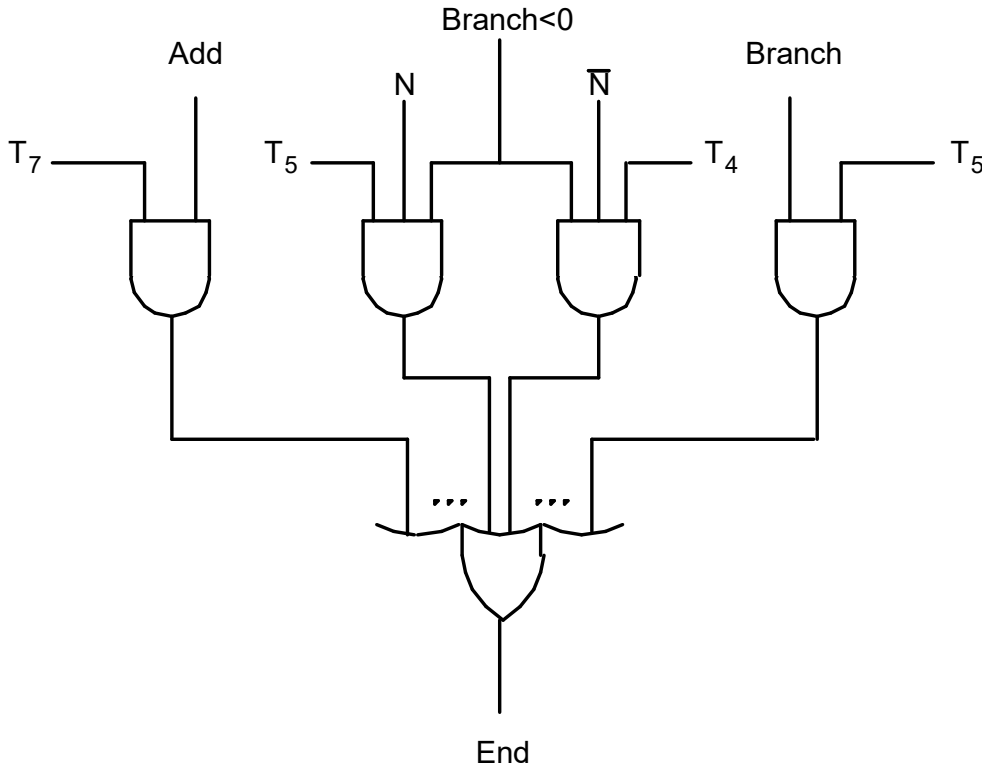


Figure 7.13. Generation of the End control signal.

A Complete Processor

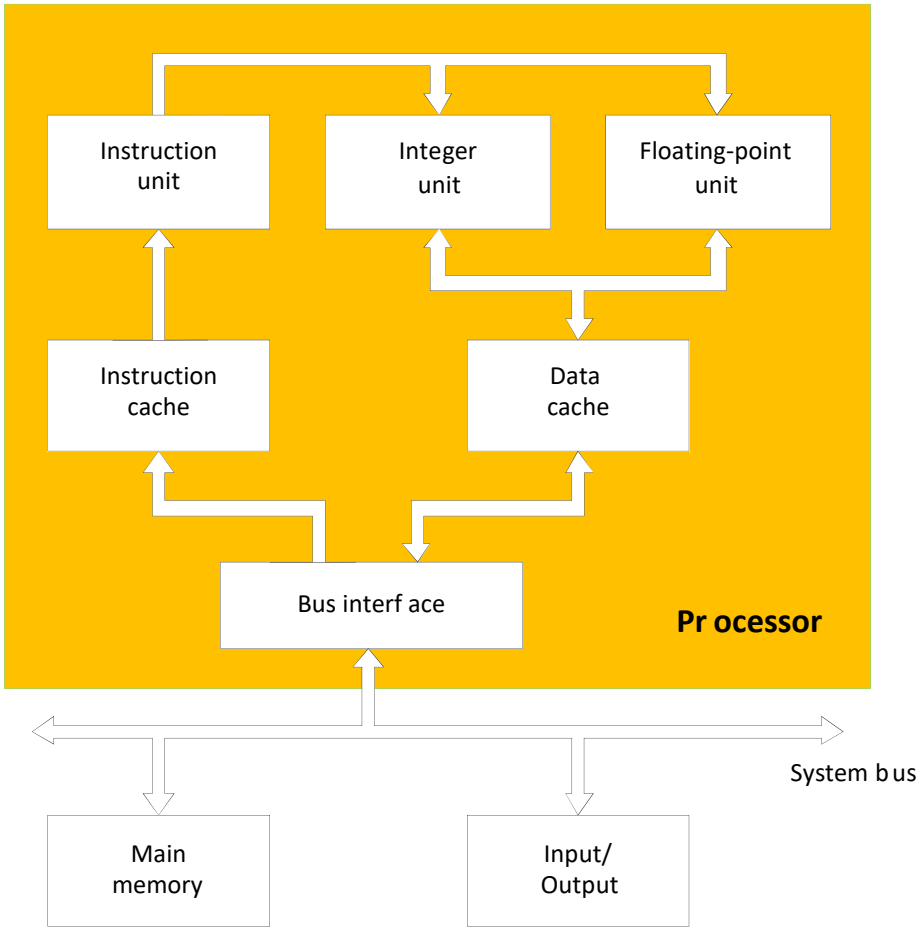
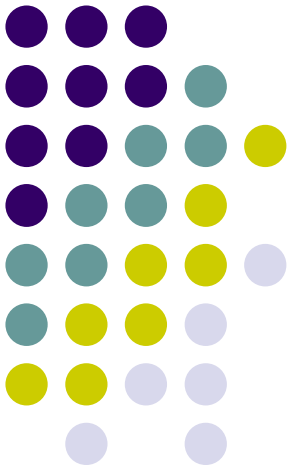
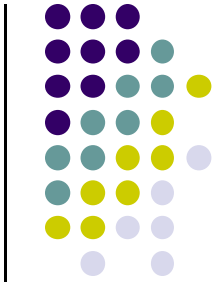


Figure 7.14. Block diagram of a complete processor

Microprogrammed Control

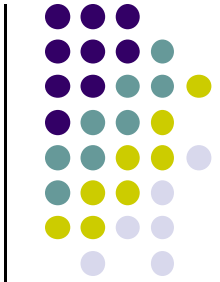


Common terms



- **Microprogrammed Control:** In which control signals are generated by a program similar to machine language programs.
- **Control Word (CW):** is a word whose individual bits represents the various control signals.
- **Microroutine:** A sequence of CW's corresponds to the control sequence of a machine instruction constitutes the microroutine for that instruction.
- **Microinstructions:** Individual control words in this microroutine are referred as microinstuctions.
- **Control Store:** The microroutines for all instructions in the instruction set of a computer are stored in a special memory.
- **Micro Program Counter (μPC):** To read the control words sequentially from the control store, a μPC is used.

Overview

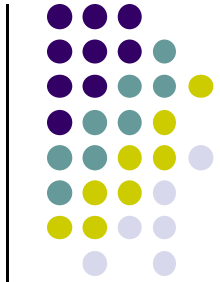


- Control signals are generated by a program similar to machine language programs.
- Control Word (CW); microroutine; microinstruction

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	

Figure 7.15 An example of microinstructions for Figure 7.6.

Overview

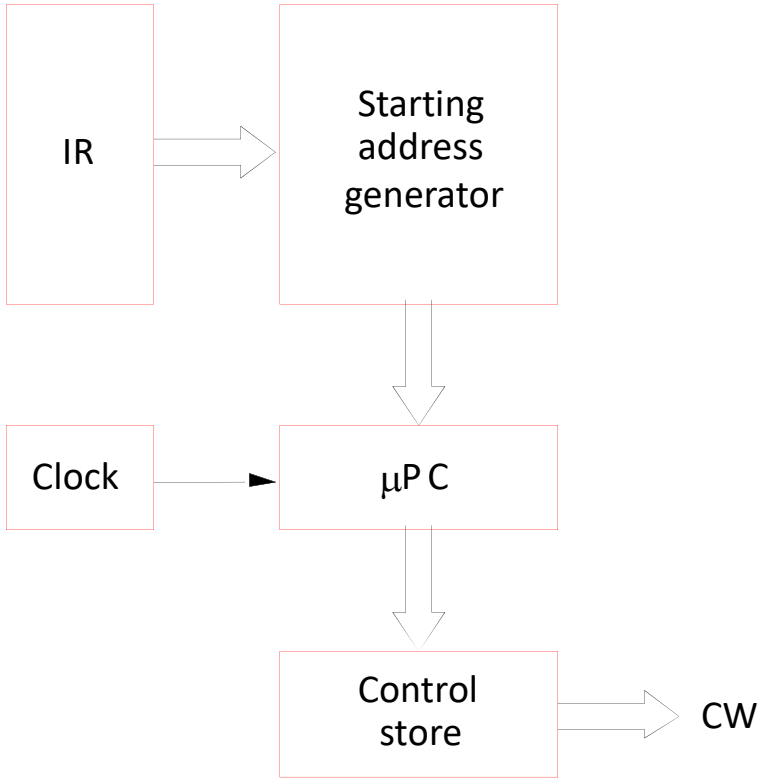


Step	Action
1	PC_{out} , MAR_{in} , Read, Select4,Add, Z_{in}
2	Z_{out} , PC_{in} , Y_{in} , WMF C
3	MDR_{out} , IR_{in}
4	$R3_{out}$, MAR_{in} , Read
5	$R1_{out}$, Y_{in} , WMF C
6	MDR_{out} , SelectY, Add, Z_{in}
7	Z_{out} , $R1_{in}$, End

Figure 7.6. Control sequence for execution of the instruction Add (R3),R1.

Overview

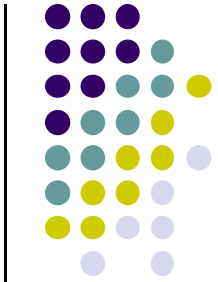
- Control store



One function cannot be carried out by this simple organization.

Figure 7.16. Basic organization of a microprogrammed control unit.

Overview



- 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.
- Use conditional branch microinstruction.

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

Figure 7.17. Microroutine for the instruction Branch<0.

Overview

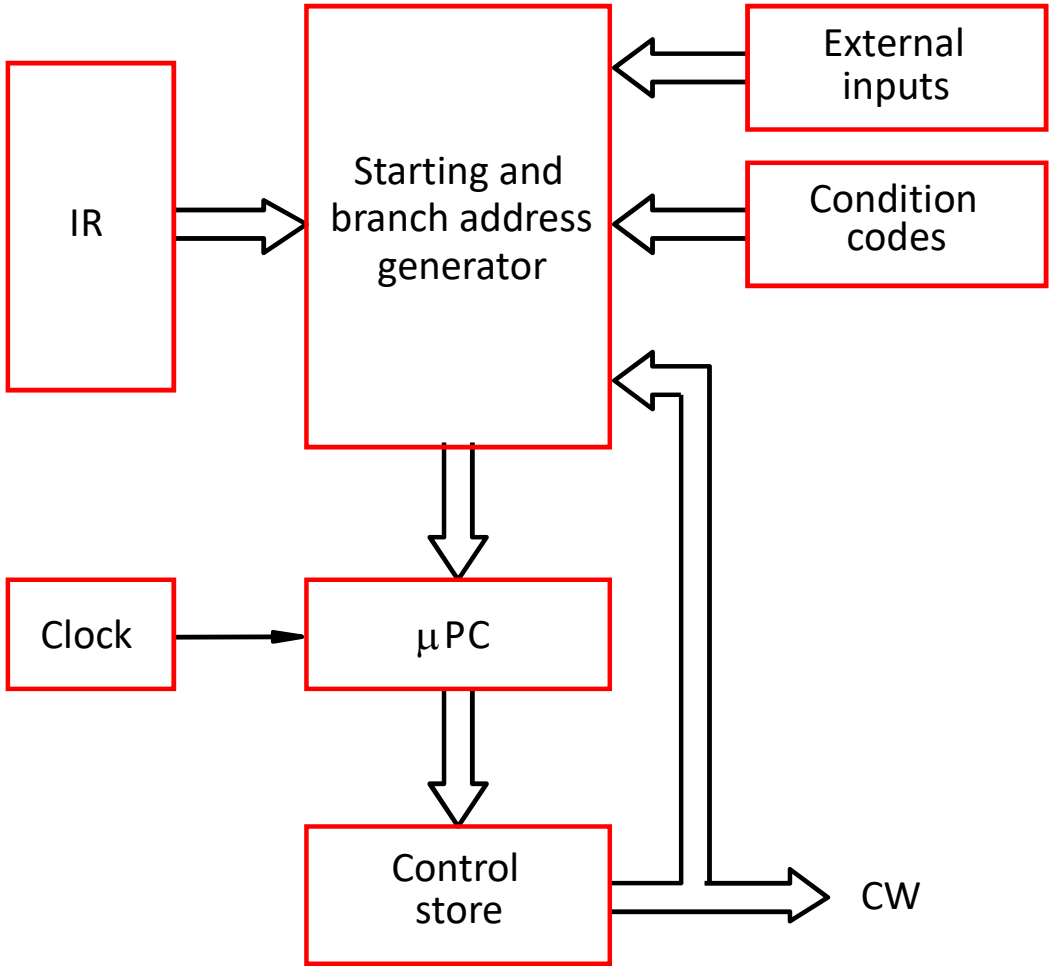
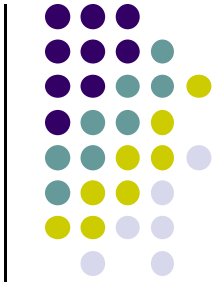


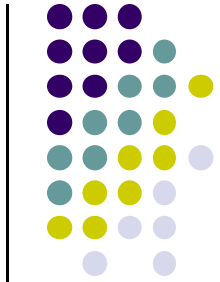
Figure 7.18. Organization of the control unit to allow conditional branching in the microprogram.

Microinstructions



- A straightforward way to structure microinstructions is to assign one bit position to each control signal.
- However, this is very inefficient.
- The length can be reduced: most signals are not needed simultaneously, and many signals are mutually exclusive.
- All mutually exclusive signals are placed in the same group in binary coding.

Partial Format for the Microinstructions



Microinstruction

F1	F2	F3	F4	F5

F1 (4 bits)	F2 (3 bits)	F3 (3 bits)	F4 (4 bits)	F5 (2 bits)
0000: No transfer	000: No transfer	000: No transfer	0000: Add	00: No action
0001: PC_{out}	001: PC_{in}	001: MAR_{in}	0001: Sub	01: Read
0010: MDR_{out}	010: IR_{in}	010: MDR_{in}	⋮	10: Write
0011: Z_{out}	011: Z_{in}	011: $TEMP_{in}$	1111: XOR	
0100: $R0_{out}$	100: $R0_{in}$	100: Y_{in}	16 ALU functions	
0101: $R1_{out}$	101: $R1_{in}$			
0110: $R2_{out}$	110: $R2_{in}$			
0111: $R3_{out}$	111: $R3_{in}$			
1010: $TEMP_{out}$				
1011: $Offset_{out}$				

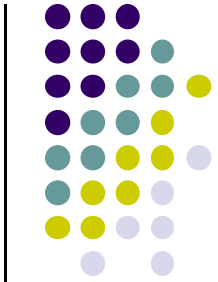
F6	F7	F8	...

F6 (1 bit)	F7 (1 bit)	F8 (1 bit)
0: SelectY 1: Select4	0: No action 1: WMFC	0: Continue 1: End

What is the price paid for this scheme?

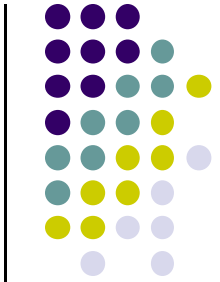
Figure 7.19. An example of a partial format for field-encoded microinstructions

Further Improvement



- Enumerate the patterns of required signals in all possible microinstructions. Each meaningful combination of active control signals can then be assigned a distinct code.
- Vertical organization
  - Highly encoded schemes that use compact codes to specify only a small number of control functions in each micro instructions.
- Horizontal organization
  - Many resource can be controlled with a single microinstructions.

Microprogram Sequencing



- If all microprograms require only straightforward sequential execution of microinstructions except for branches, letting a μPC governs 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.
- Example: Add src, Rdst
- Four addressing modes: register, autoincrement, autodecrement, and indexed (with indirect forms).

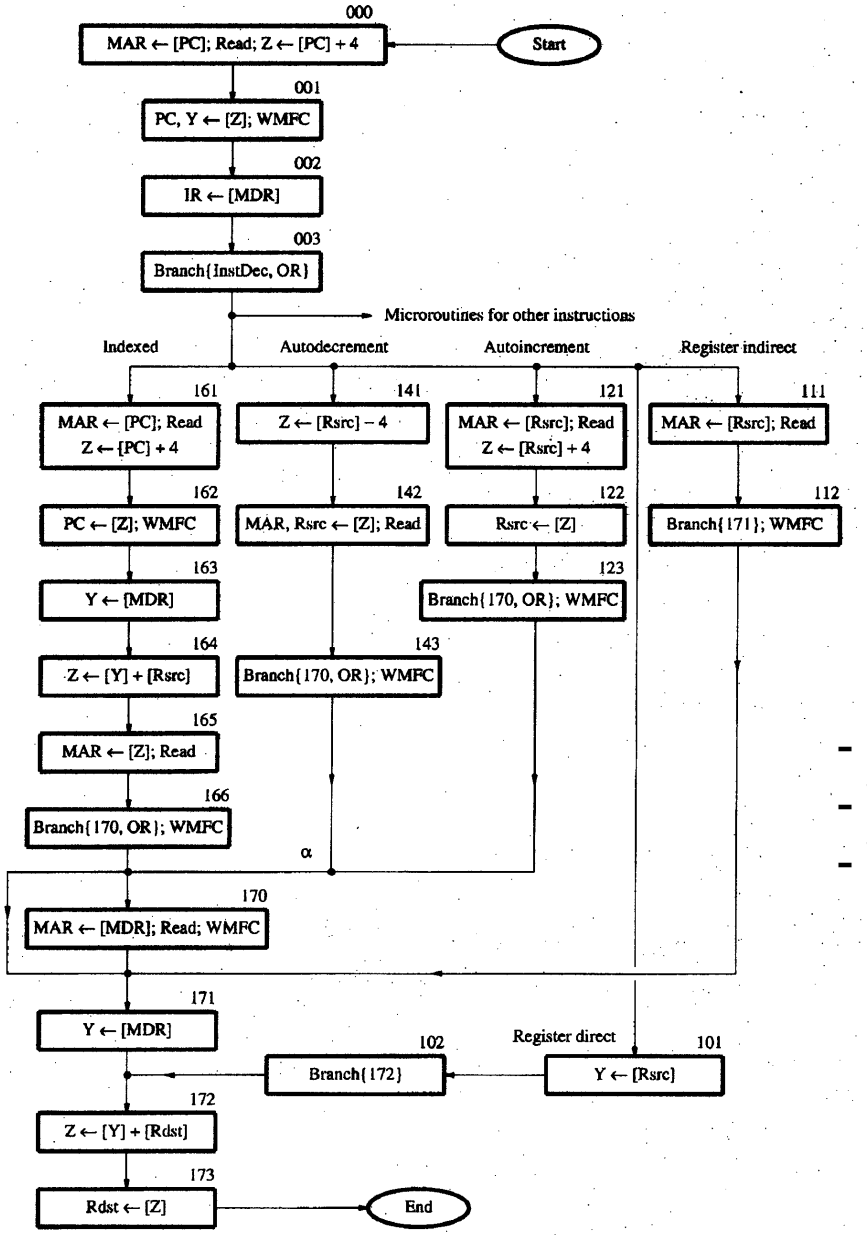
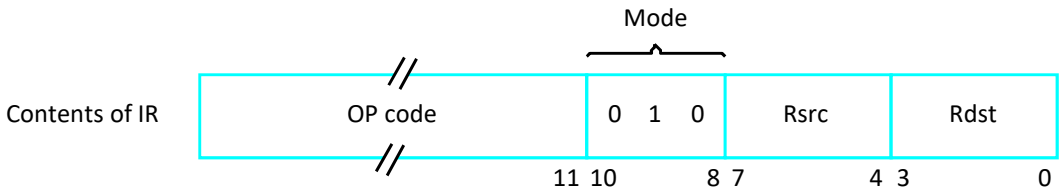


Figure 7.20. Flowchart of a microprogram for the Add src,Rdst instruction.

- Bit-ORing
- Wide-Branch Addressing
- WMFC

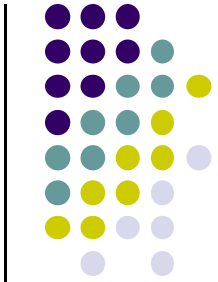


Address (octal)	Microinstruction
000	PC_{out}, MAR_{in}, Read, Select4, Add, Z_{in}
001	Z_{out}, PC_{in}, Y_{in}, WMFC
002	MDR_{out}, IR_{in}
003	μBranch {μPC $\leftarrow$ 101 (from Instruction decoder); $\mu PC_{5,4} \leftarrow [IR_{10,9}]$; $\mu PC_3 \leftarrow [\overline{IR_{10}}] \cdot [\overline{IR_9}] \cdot [IR_8]$}
121	$Rsrc_{out}$, MAR_{in}, Read, Select4, Add, Z_{in}
122	Z_{out}, $Rsrc_{in}$
123	μBranch {μPC $\leftarrow$ 170; $\mu PC_0 \leftarrow [\overline{IR_8}]$}, WMFC
170	MDR_{out}, MAR_{in}, Read, WMFC
171	MDR_{out}, Y_{in}
172	$Rdst_{out}$, SelectY, Add, Z_{in}
173	Z_{out}, $Rdst_{in}$, End

Figure 7.21. Microinstruction for Add (Rsrc)+,Rdst.

Note: Microinstruction at location 170 is not executed for this addressing mode.

Microinstructions with Next-Address Field



- The microprogram we discussed requires several branch microinstructions, which perform no useful operation in the datapath.
- A powerful alternative approach is to 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 for the address field (around 1/6)

Microinstructions with Next-Address Field

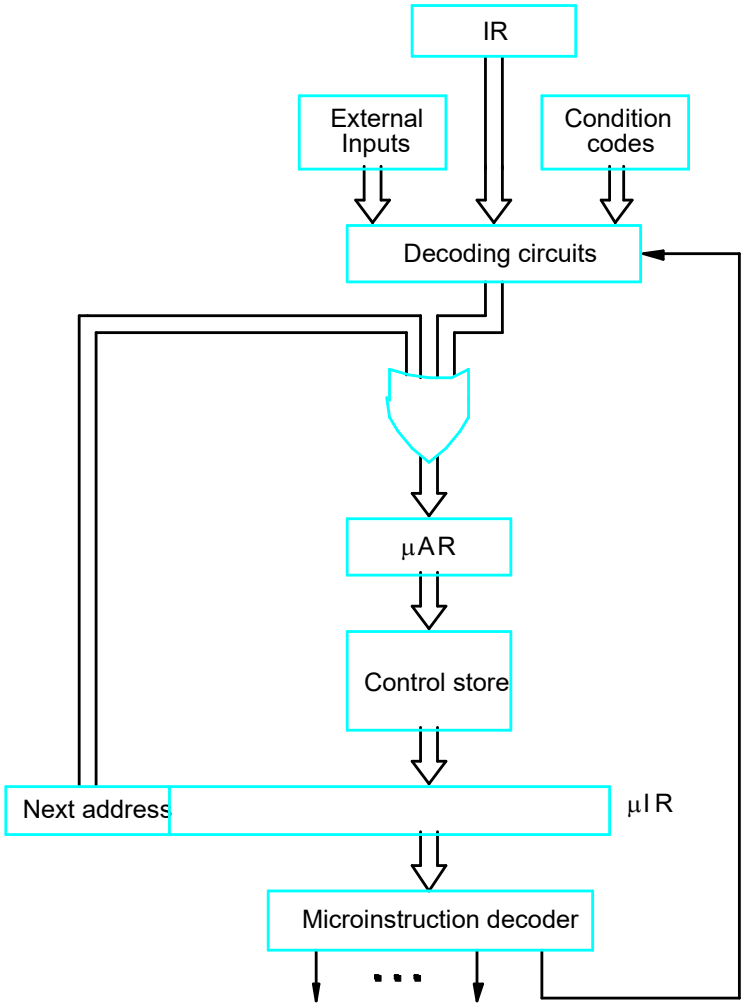


Figure 7.22. Microinstruction-sequencing organization.

Microinstruction

F0	F1	F2	F3

F0 (8 bits)	F1 (3 bits)	F2 (3 bits)	F3 (3 bits)
Address of next microinstruction	000: No transfer 001: PC_{out} 010: MDR_{out} 011: Z_{out} 100: $Rsrc_{out}$ 101: $Rdst_{out}$ 110: $TEMP_{out}$	000: No transfer 001: PC_{in} 010: IR_{in} 011: Z_{in} 100: $Rsrc_{in}$ 101: $Rdst_{in}$	000: No transfer 001: MAR_{in} 010: MDR_{in} 011: $TEMP_{in}$ 100: Y_{in}

F4	F5	F6	F7

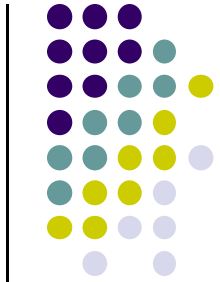
F4 (4 bits)	F5 (2 bits)	F6 (1 bit)	F7 (1 bit)
0000: Add 0001: Sub ⋮ 1111: XOR	00: No action 01: Read 10: Write	0: SelectY 1: Select4	0: No action 1: WMFC

F8	F9	F10

F8 (1 bit)	F9 (1 bit)	F10 (1 bit)
0: NextAdrs 1: InstDec	0: No action 1: OR_{mode}	0: No action 1: OR_{indsrc}

Figure 7.23. Format for microinstructions in the example of Section 7

Implementation of the Microroutine



Octal address	F0	F1	F2	F3	F4	F5	F6	F7	F8	F9	F10
0 0 0	0 0 0 0 0 0 0 1	0 0 1	0 1 1	0 0 1	0 0 0 0	0 1	1	0	0	0	0
0 0 1	0 0 0 0 0 0 1 0	0 1 1	0 0 1	1 0 0	0 0 0 0	0 0	0	1	0	0	0
0 0 2	0 0 0 0 0 0 1 1	0 1 0	0 1 0	0 0 0	0 0 0 0	0 0	0	0	0	0	0
0 0 3	0 0 0 0 0 0 0 0	0 0 0	0 0 0	0 0 0	0 0 0 0	0 0	0	0	1	1	0
1 2 1	0 1 0 1 0 0 1 0	1 0 0	0 1 1	0 0 1	0 0 0 0	0 1	1	0	0	0	0
1 2 2	0 1 1 1 1 0 0 0	0 1 1	1 0 0	0 0 0	0 0 0 0	0 0	0	1	0	0	1
1 7 0	0 1 1 1 1 0 0 1	0 1 0	0 0 0	0 0 1	0 0 0 0	0 1	0	1	0	0	0
1 7 1	0 1 1 1 1 0 1 0	0 1 0	0 0 0	1 0 0	0 0 0 0	0 0	0	0	0	0	0
1 7 2	0 1 1 1 1 0 1 1	1 0 1	0 1 1	0 0 0	0 0 0 0	0 0	0	0	0	0	0
1 7 3	0 0 0 0 0 0 0 0	0 1 1	1 0 1	0 0 0	0 0 0 0	0 0	0	0	0	0	0

Figure 7.24. Implementation of the microroutine of Figure 7.21 usi
next-microinstruction address fie(See Figure 7.23 for encoded sign

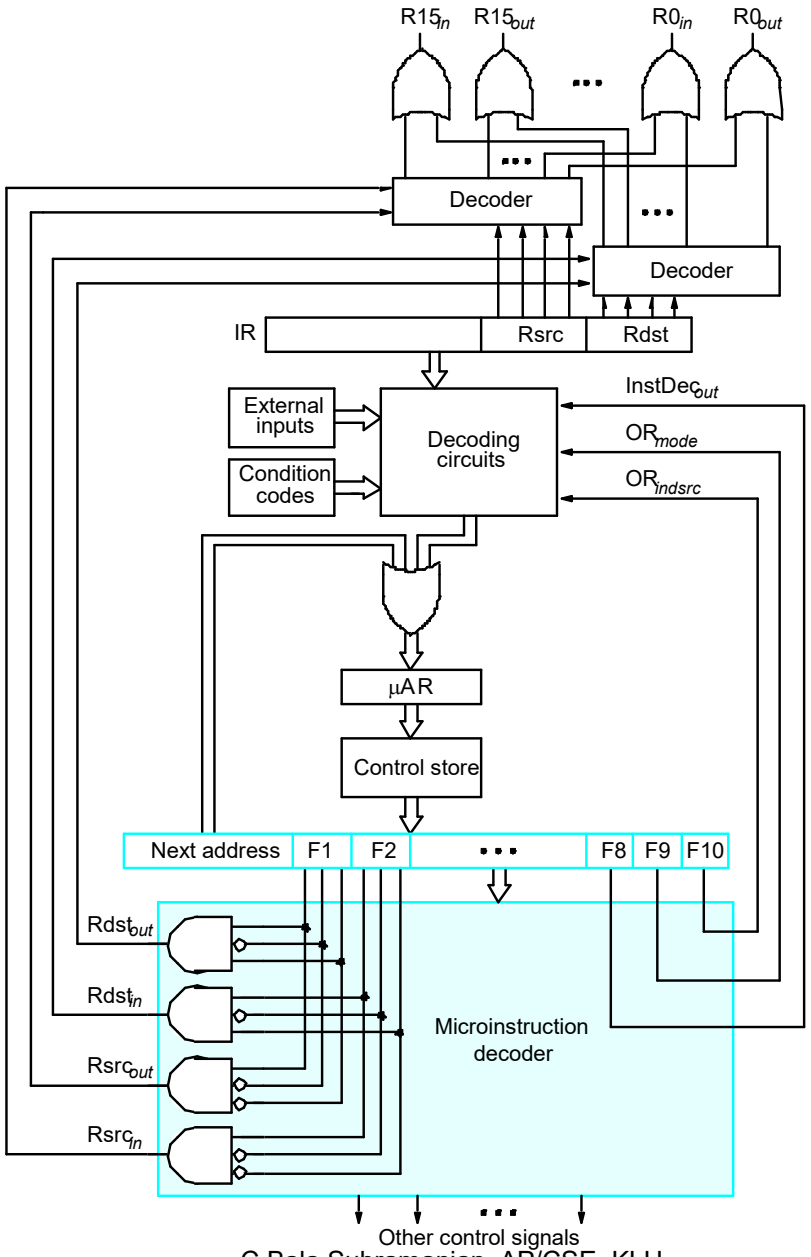


Figure 7.25. Some details of the control-signal-generating circuitry.

bit-ORing

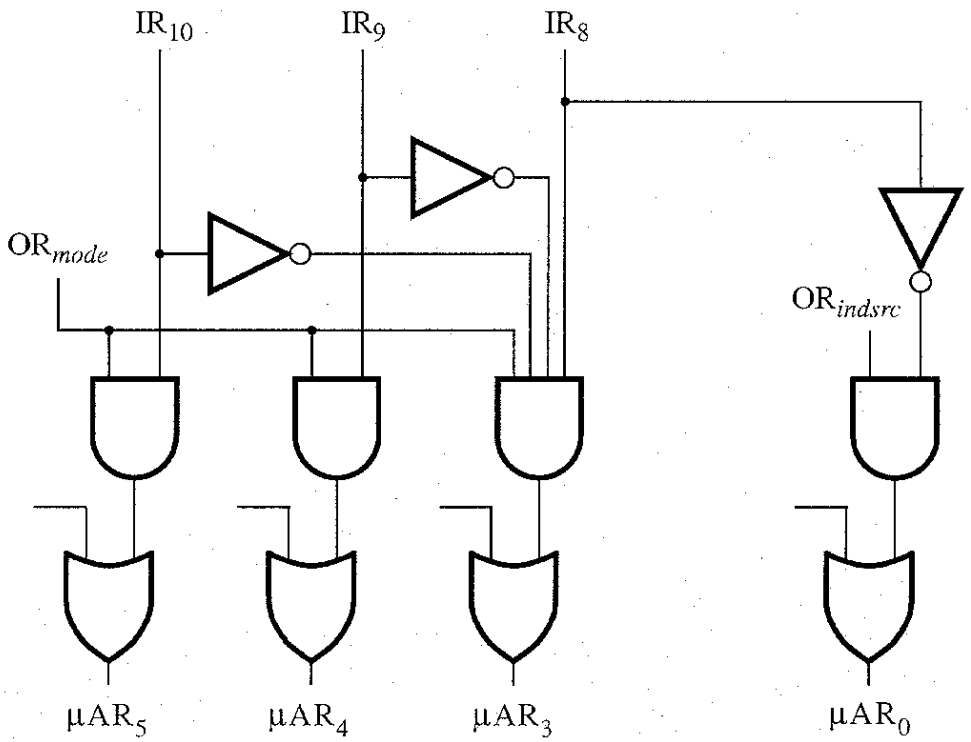
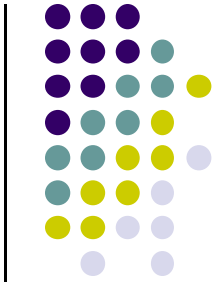


Figure 7.26. Control circuitry for bit-ORing

(part of the decoding circuits in Figure 7.25).

Further Discussions



- Prefetching
- Emulation