Chapter 7
Basic processing Unit

Chapter Objectives

- How a processor executes instructions
- Internal functional units and how they are connected
- Hardware for generating internal control signals
- The micro programming approach
- Micro program organization

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

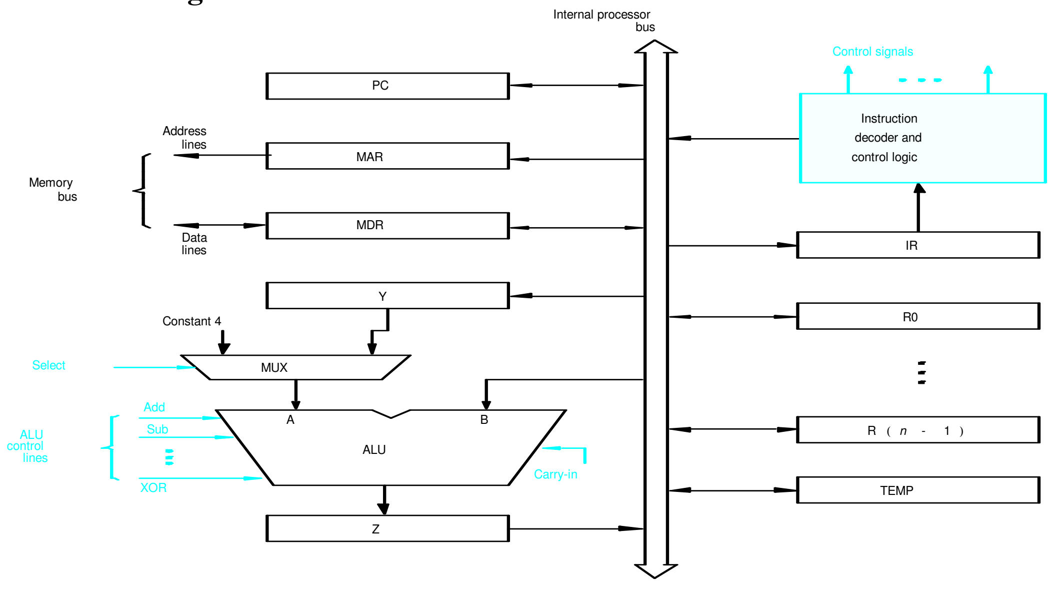

Figure 7.1. Single-bus organization of the datapath inside a processor.

- ALU and all the registers are interconnected via a single common bus.
- The data and address lines of the external memory bus connected to the internal processor bus via the memory data register, MDR, and the memory address register, MAR respectively.
- Register MDR has two inputs and two outputs.
- Data may be loaded into MDR either from the memory bus or from the internal processor bus.
- The data stored in MDR may be placed on either bus.
- The input of MAR is connected to the internal bus, and its output is connected to the external bus.
- The control lines of the memory bus are connected to the instruction decoder and control logic.
- This unit is responsible for issuing the signals that control the operation of all the units inside the processor and for increasing with the memory bus.
- The MUX selects either the output of register Y or a constant value 4 to be provided as input A of the ALU.
- The constant 4 is used to increment the contents of the program counter.

Register Transfers

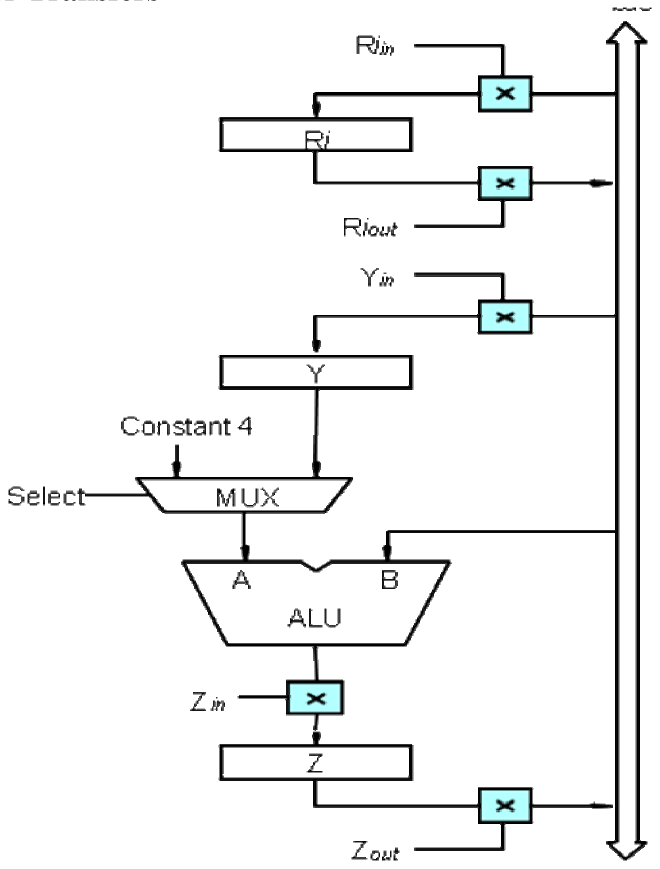

Figure 7.2. Input and output gating for the registers in Figure 7.1.

- Instruction execution involves a sequence of steps in which data are transferred from one register to another.
- For each register two control signals are used to place the contents of that register on the bus or to load the data on the bus into register.(in figure)
- The input and output of register Ri_{in} and Ri_{out} is set to 1, the data on the bus are loaded into R_i.
- Similarly, when Ri $_{out}$ is set to 1, the contents of register Ri are placed on the bus.
- While Ri_{out} is equal to 0, the bus can be used for transferring data from other registers.

Example

- Suppose we wish to transfer the contents of register R1 to register R4. This can be accomplished as follows.
- Enable the output of registers R1 by setting R1out to 1. This places the contents of R1 on the processor bus.
- Enable the input of register R4 by setting R4out to 1. This loads data from the processor bus into register R4.
- All operations and data transfers with in the processor take place with in time periods defined by the processor clock.
- The control signals that govern a particular transfer are asserted at the start of the clock cycle.

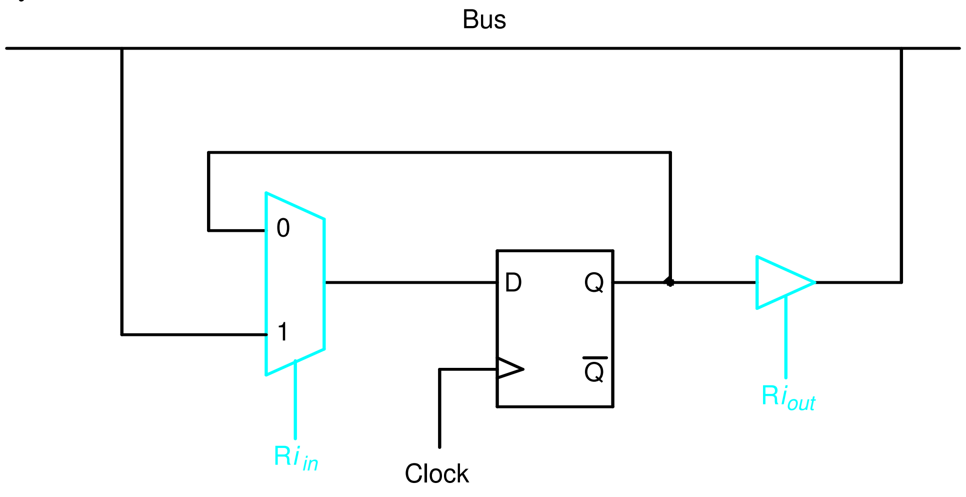

Figure 7.3. Input and output g ating for one register bit.

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?
    - R1out, Yin

    - R2out, SelectY, Add, Zin
    - Zout, R3in
- All other signals are inactive.
- In step 1, the output of register R1 and the input of register Y are enabled, causing the contents of R1 to be transferred over the bus to Y.
- Step 2, the multiplexer's select signal is set to Select Y, causing the multiplexer to gate the contents of register Y to input A of the ALU.
- At the same time, the contents of register R2 are gated onto the bus and, hence, to input B.
- The function performed by the ALU depends on the signals applied to its control lines.
- In this case, the ADD line is set to 1, causing the output of the ALU to be the sum of the two numbers at inputs A and B.
- This sum is loaded into register Z because its input control signal is activated.
- In step 3, the contents of register Z are transferred to the destination register R3. This last transfer cannot be carried out during step 2, because only one register output can be connected to the bus during any clock cycle.

Fetching a Word from Memory

- The processor has to specify the address of the memory location where this information is stored and request a Read operation.
- This applies whether the information to be fetched represents an instruction in a program or an operand specified by an instruction.
- The processor transfers the required address to the MAR, whose output is connected to the address lines of the memory bus.

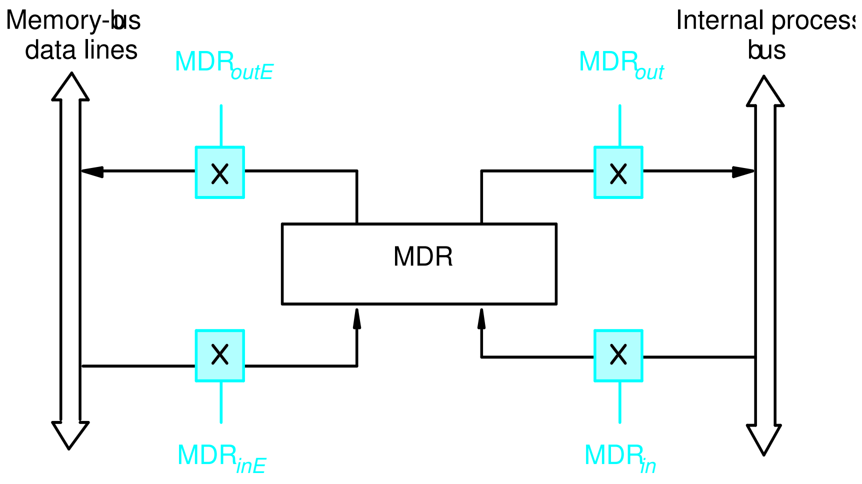

Figure 7.4. Connection and control signals for register MDR.

- At the same time , the processor uses the control lines of the memory bus to indicate that a Read operation is needed.
- When the requested data are received from the memory they are stored in register MDR, from where they can be transferred to other registers in the processor.

- 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]
- The output of MAR is enabled all the time.
- Thus the contents of MAR are always available on the address lines of the memory bus.
- When a new address is loaded into MAR, it will appear on the memory bus at the beginning of the next clock cycle.(in fig)
- A read control signal is activated at the same time MAR is loaded.
- This means memory read operations requires three steps, which can be described by the signals being activated as follows

  $R1_{out}$,MAR_{in},Read
  MDR_{inE},WMFC
  MDR_{out},$R2_{in}$

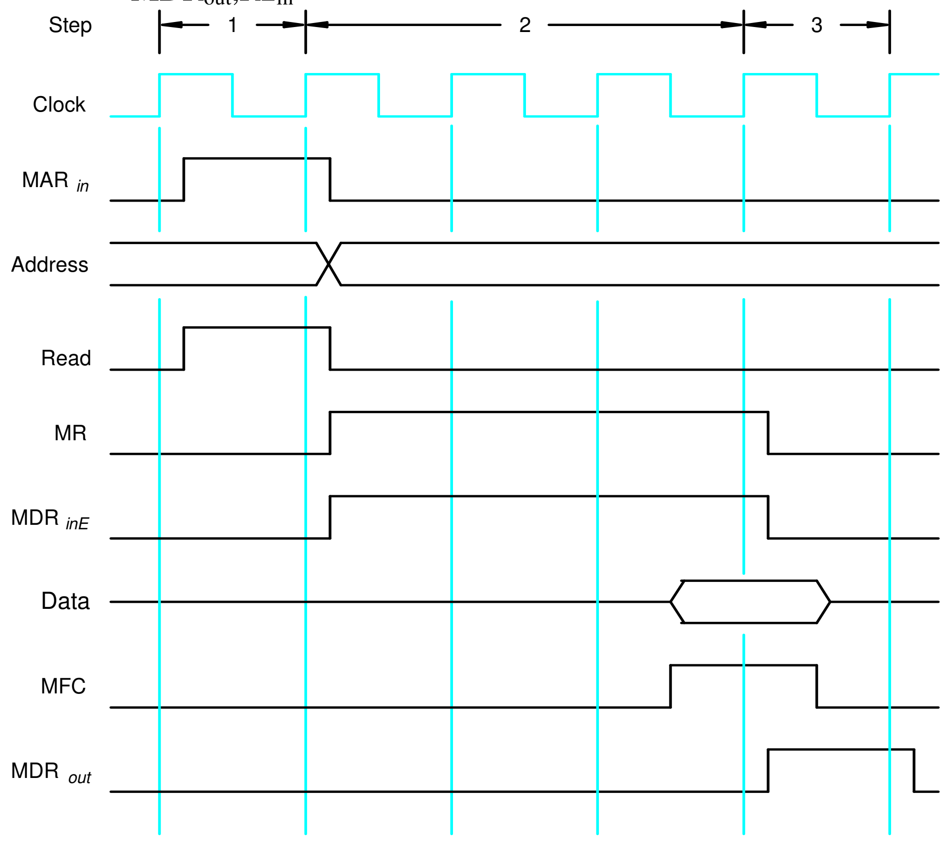

Figure 7.5. Timing of a memory Read operation.

Storing a word in Memory

- Writing a word into a memory location follows a similar procedure.
- The desired address is loaded into MAR.
- Then , the data to be written are loaded into MDR, and a write command is issued.

Example

- Executing the instruction
- Move R2,(R1) requires the following steps
  - **1 $R1_{out}$,MAR_{in}**
  - **2.$R2_{out}$,MDR_{in},Write**
  - **3.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

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.

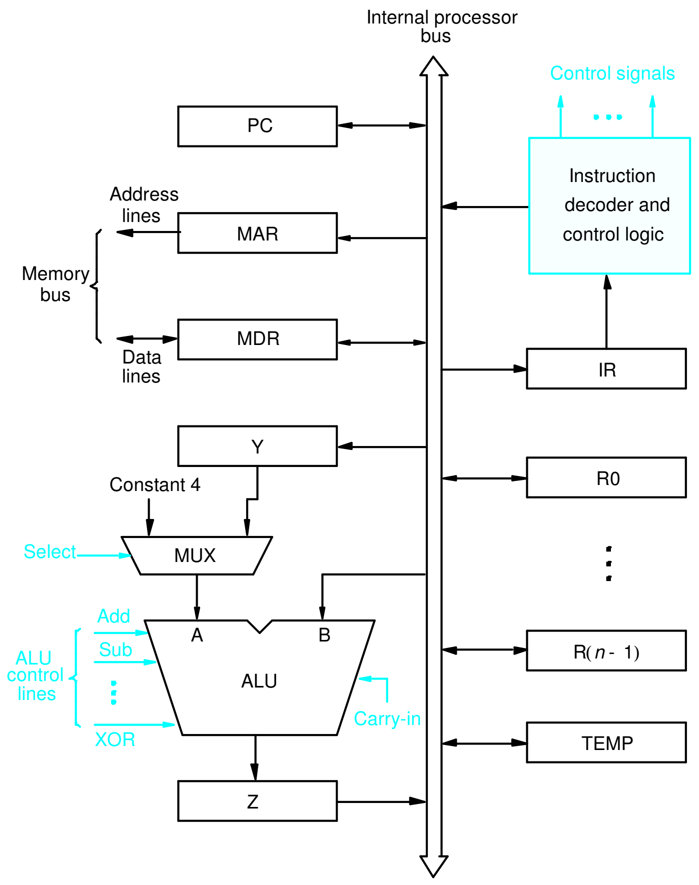

Figure 7.1. Single-bus organization of the datapath inside a proc

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

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}, Add, Z_{in}
5	Z_{out}, PC_{in}, End

Figure 7.7. Control sequence for an unconditional branch instruction.

Multiple-Bus Organization

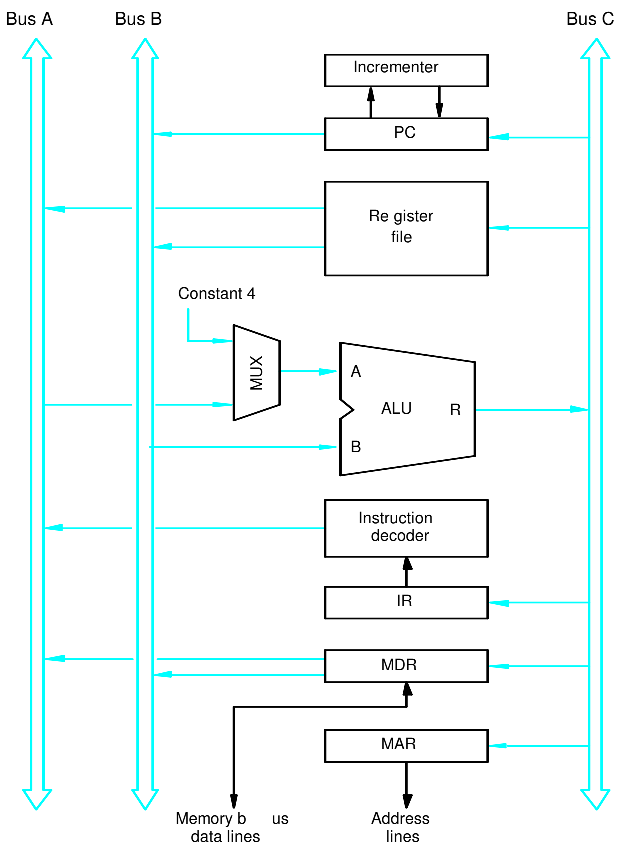

Figure 7.8. Three-bus organization of the datapath.

Example : 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.

Hardwired Control

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

Control Unit Organization

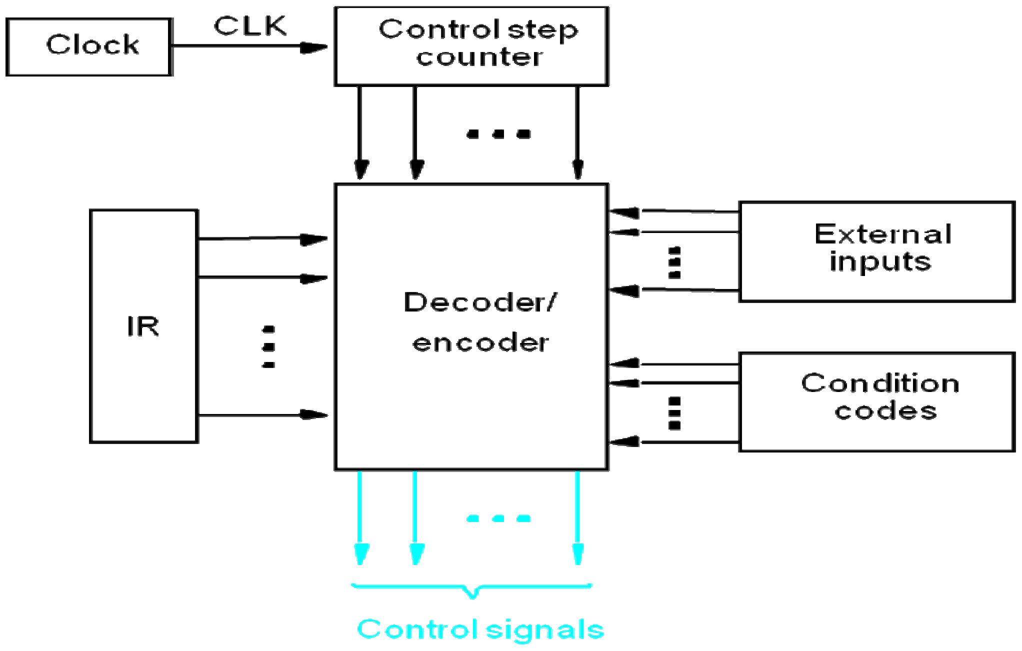

Figure 7.10. Control unit organization.

n

Detailed Control design

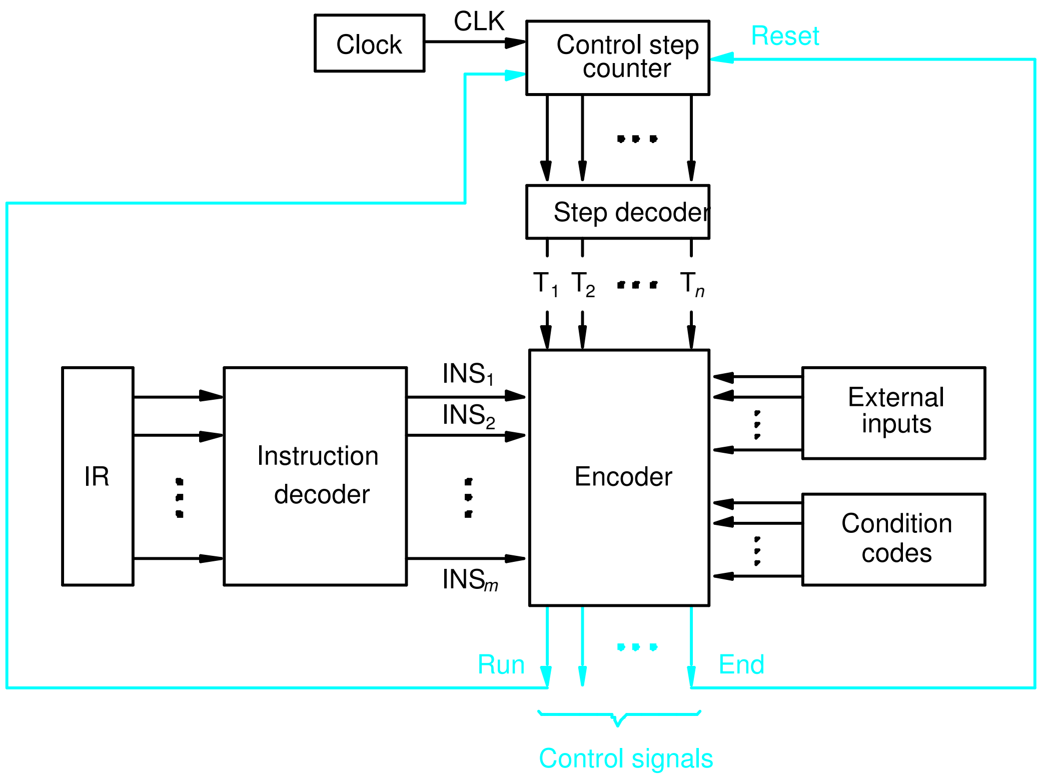

Figure 7.11. Separation of the decoding and encoding functio

Generating Z_{in}

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

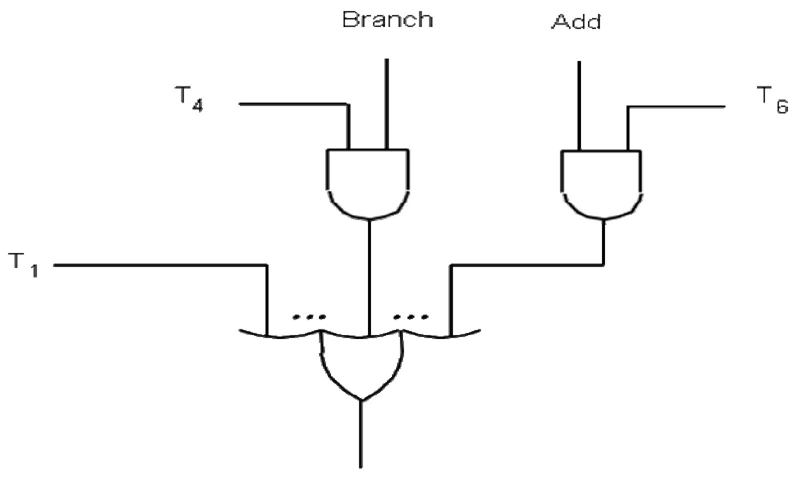

Figure 7.12. Generation of the Z_{in} control signal for the processor in Figure 7.1.

Generating End

- **End = T_7 • ADD + T_5 • BR + (T_5 • N + T_4 • N) • BRN +…**

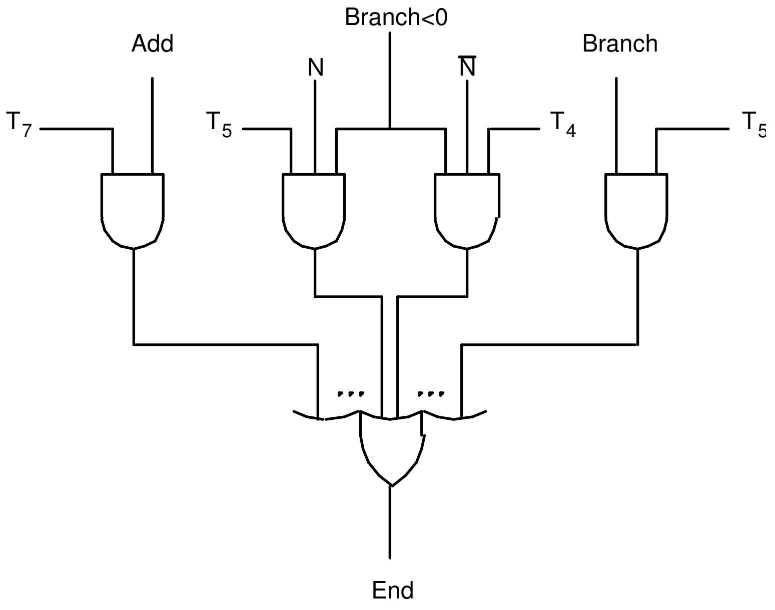

Figure 7.13. Generation of the End control signal.

A Complete Processor

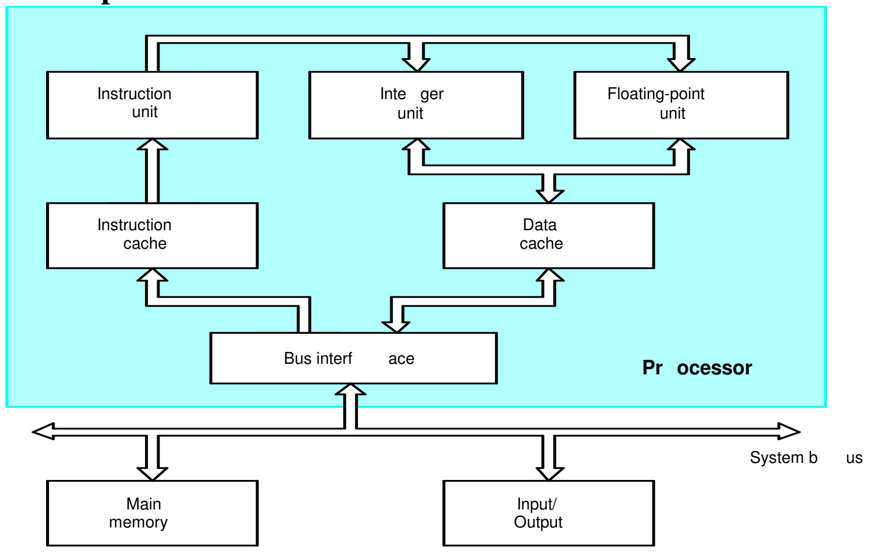

Figure 7.14. Block diagram of a complete processor .

Microprogrammed Control

- 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

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.

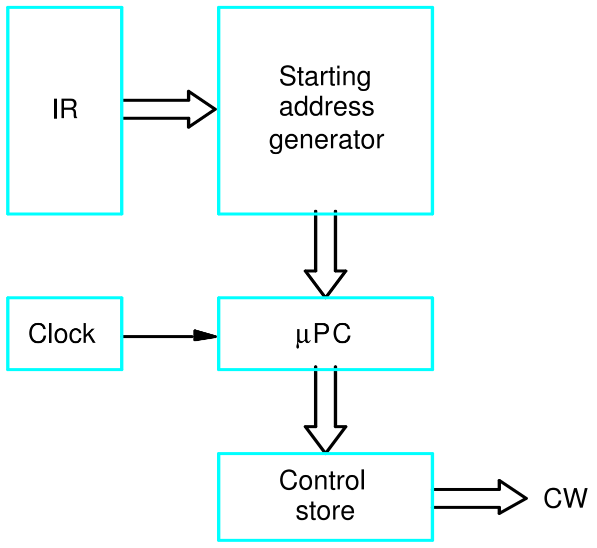

Figure 7.16. Basic organization of a microprogrammed control unit.

- 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} , WMFC
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

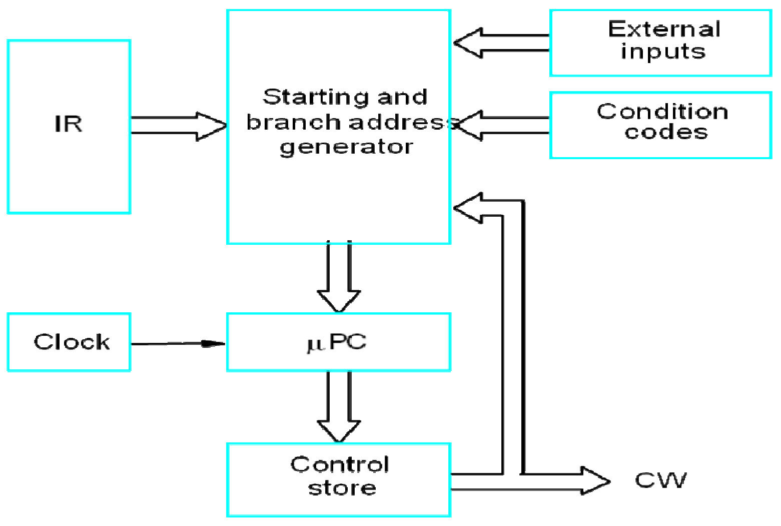

Figure 7.18. Organization of the control unit to allow conditional branching in the microprogram.

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.

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	0: No action	0: Continue
1: Select4	1: WMFC	1: End

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
- Horizontal organization

Micro program Sequencing

- If all micro programs 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 micro routine 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).

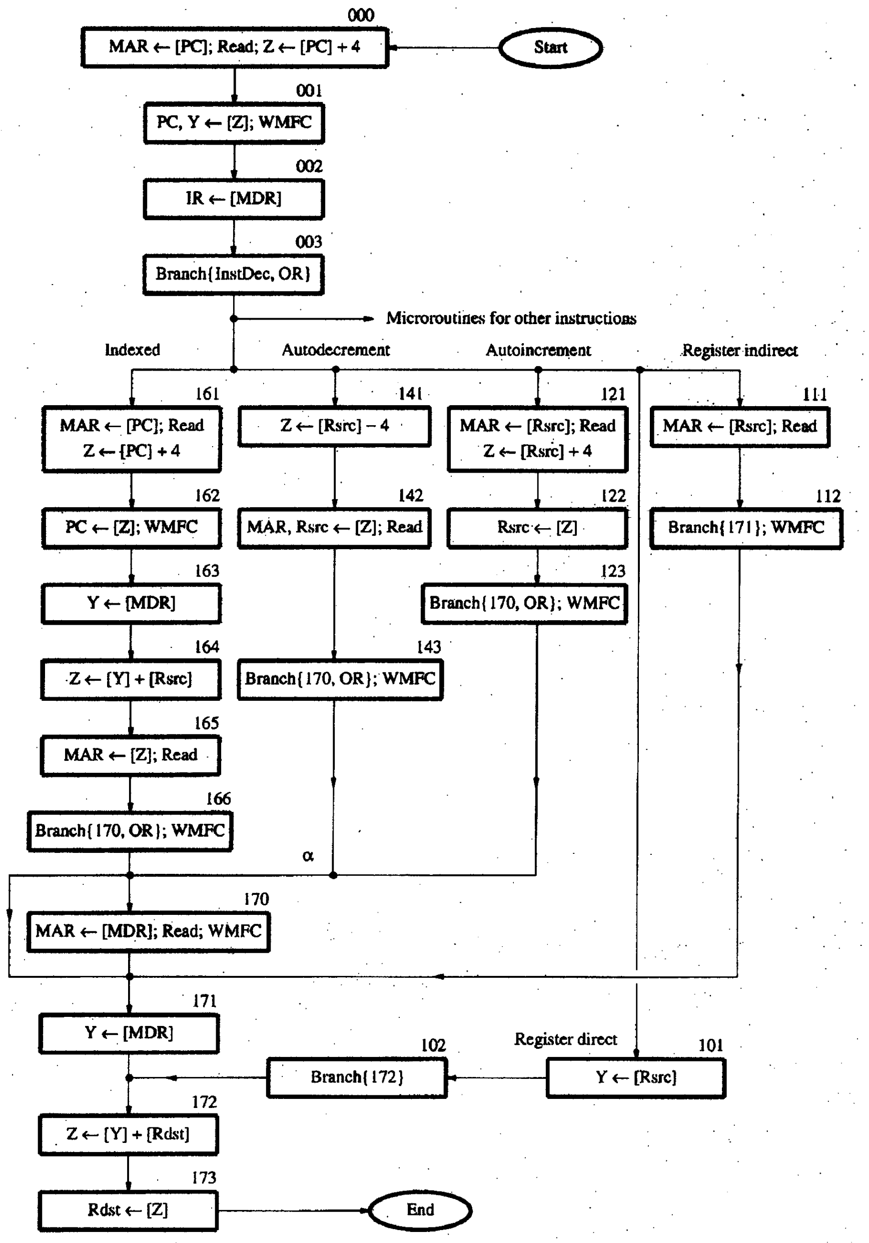

Figure 7.20. Flowchart of a microprogram for the Add src,Rdst instruction.

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 {$\mu PC \leftarrow$ 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 {$\mu PC \leftarrow$ 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.

Microinstructions with Next-Address Field

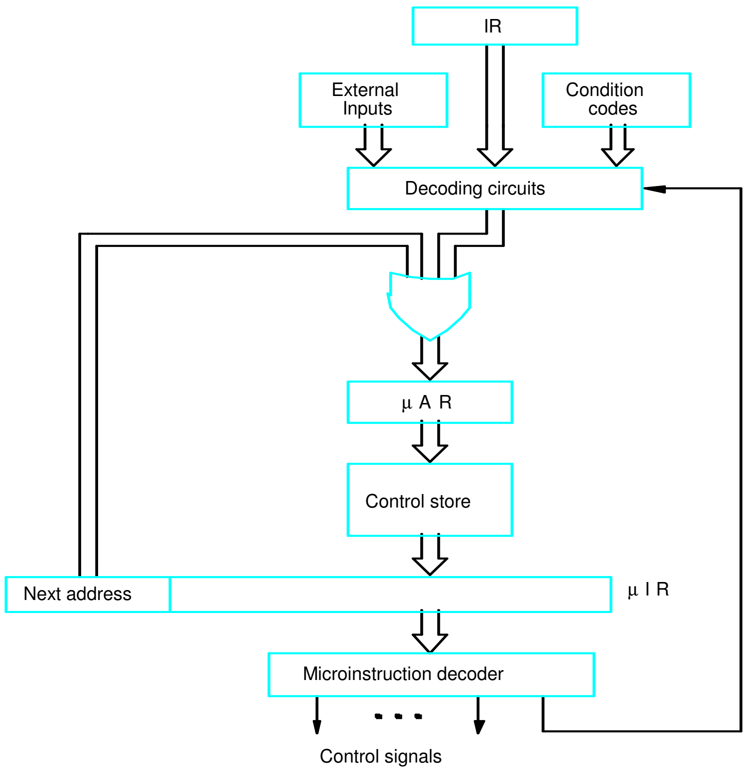

Figure 7.22. Microinstruction-sequencing organization.

- 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)

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 using a next-microinstruction address field. (See Figure 7.23 for encoded signals.)

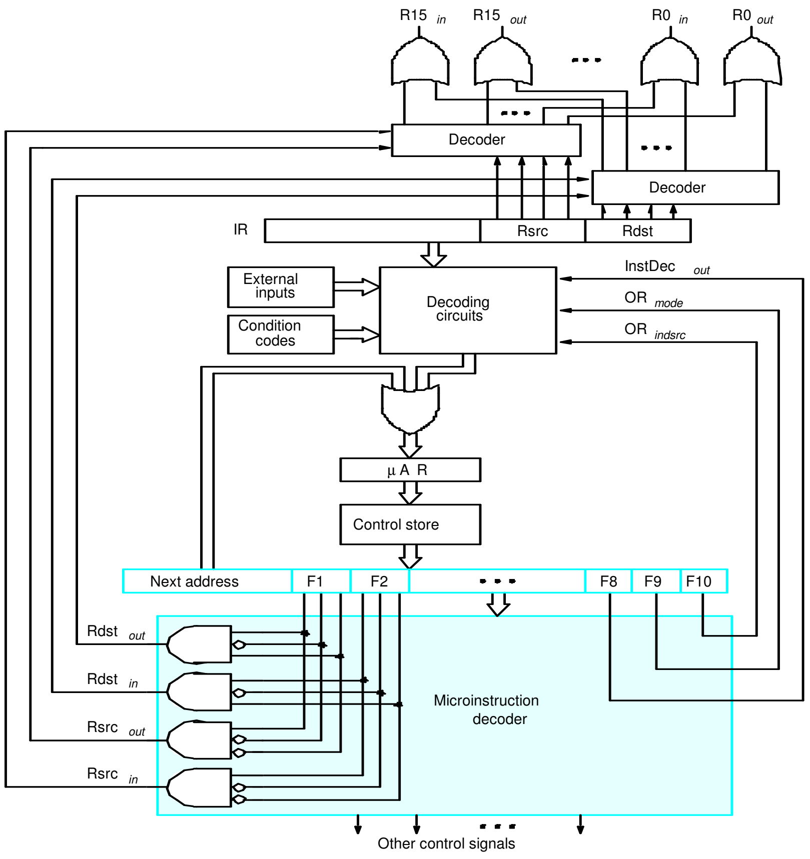

Figure 7.25. Some details of the control-signal-generating circuitry.

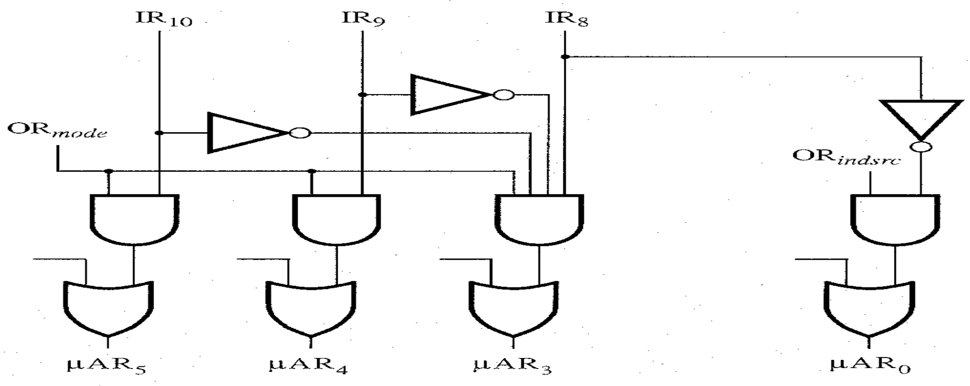

Figure 7.26. Control circuitry for bit-ORing (part of the decoding circuits in Figure 7.25).

Further Discussions

- Prefetching
- Emulation