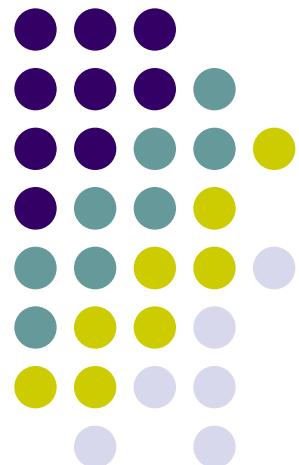


Chapter 7. Basic Processing Unit

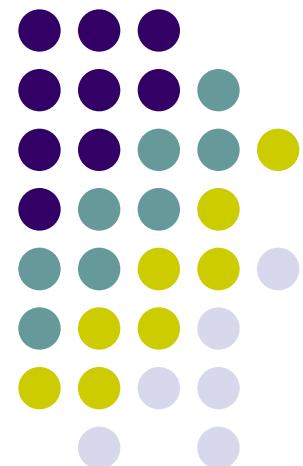




Overview

- 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





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

MDR HAS
TWO INPUTS
AND TWO
OUTPUTS

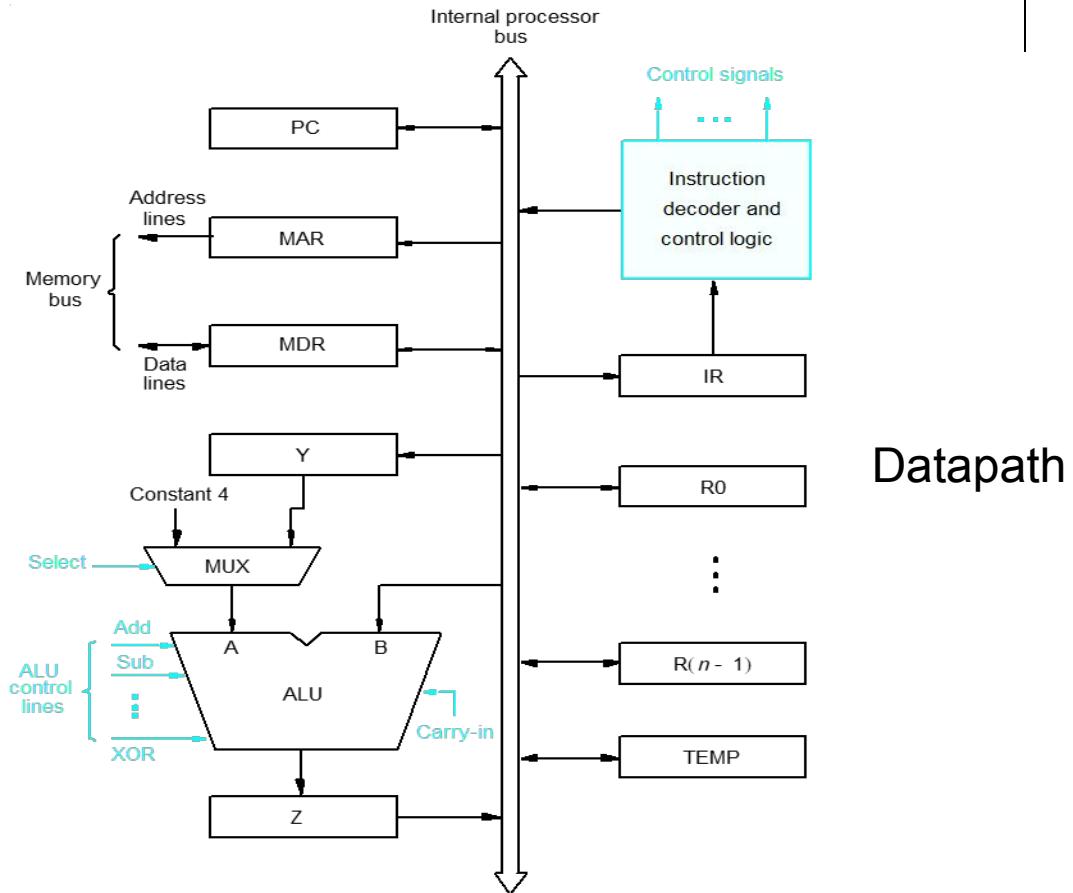


Figure 7.1. Single-bus organization of the datapath inside a processor



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.



Register Transfers

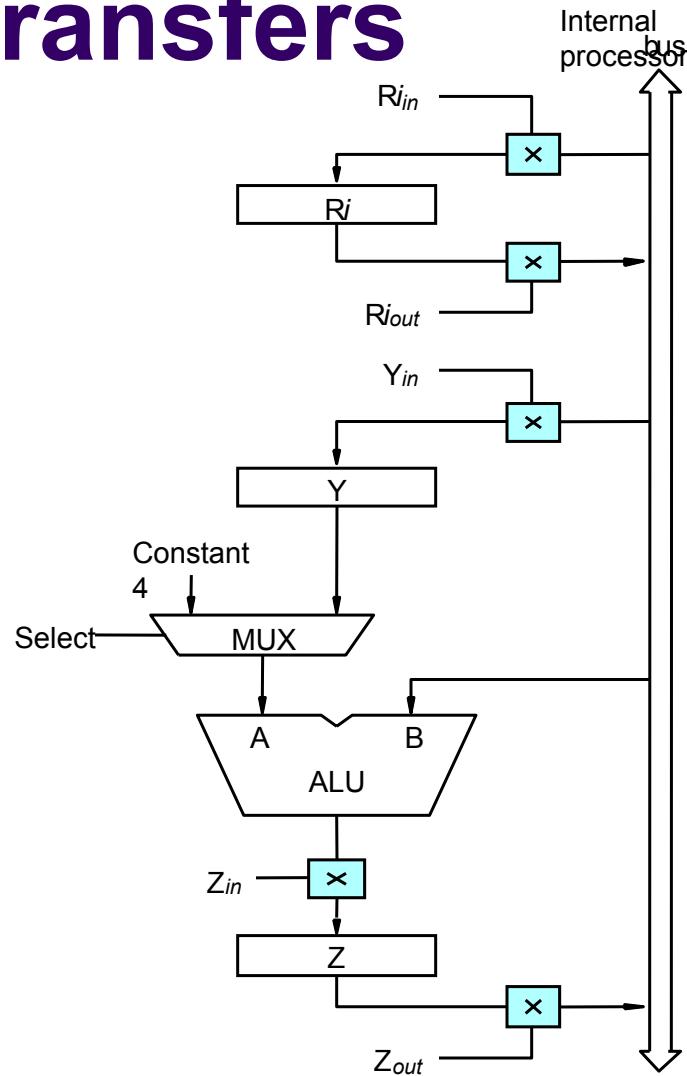
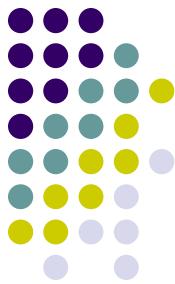


Figure 7.2. Input and output gating for the registers in Figure 7.1.

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



Fetching a Word from Memory

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

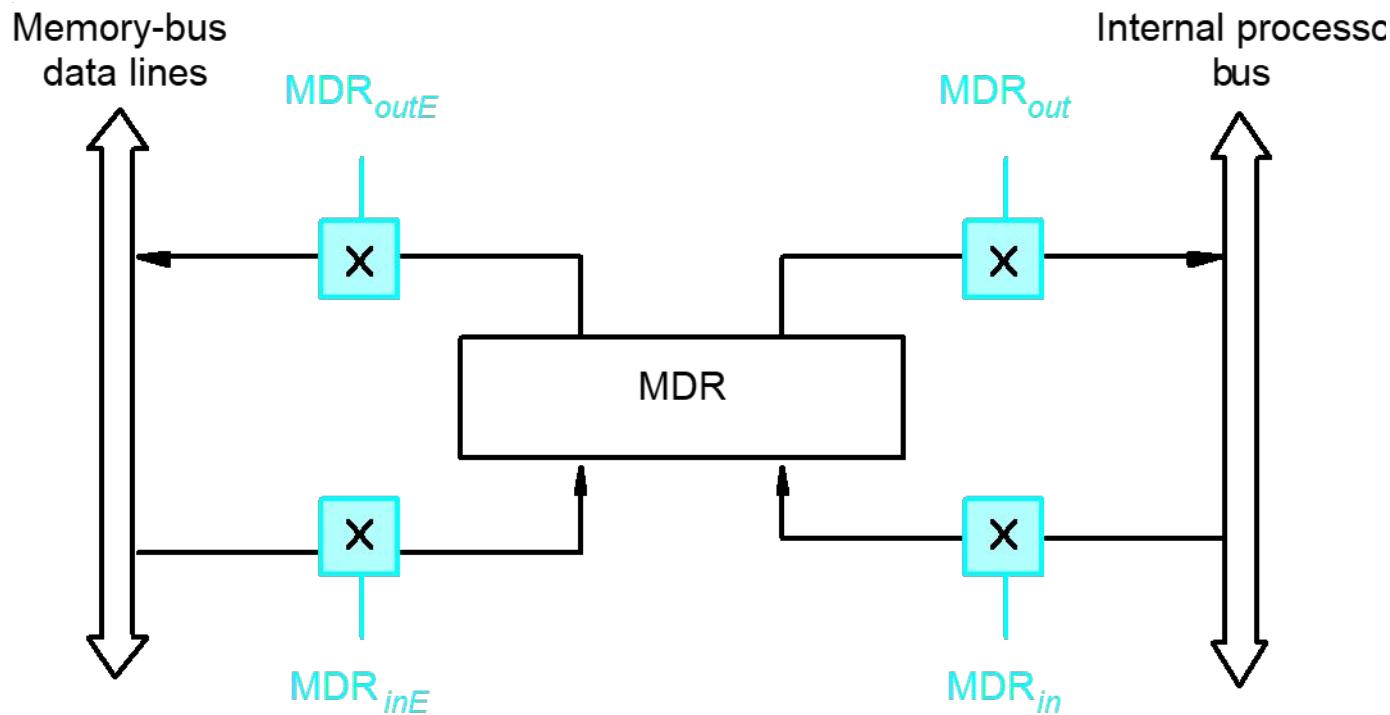


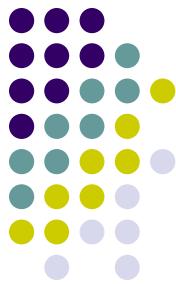
Figure 7.4. Connection and control signals for register MDR.



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 R2, (R1)
 - MAR \leftarrow [R1]
 - Start a Read operation on the memory bus
 - Wait for the MFC response from the memory
 - Load MDR from the memory bus
 - R2 \leftarrow [MDR]

Execution of a Complete Instruction



- Add R1,(R3)
- 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

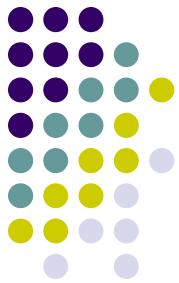
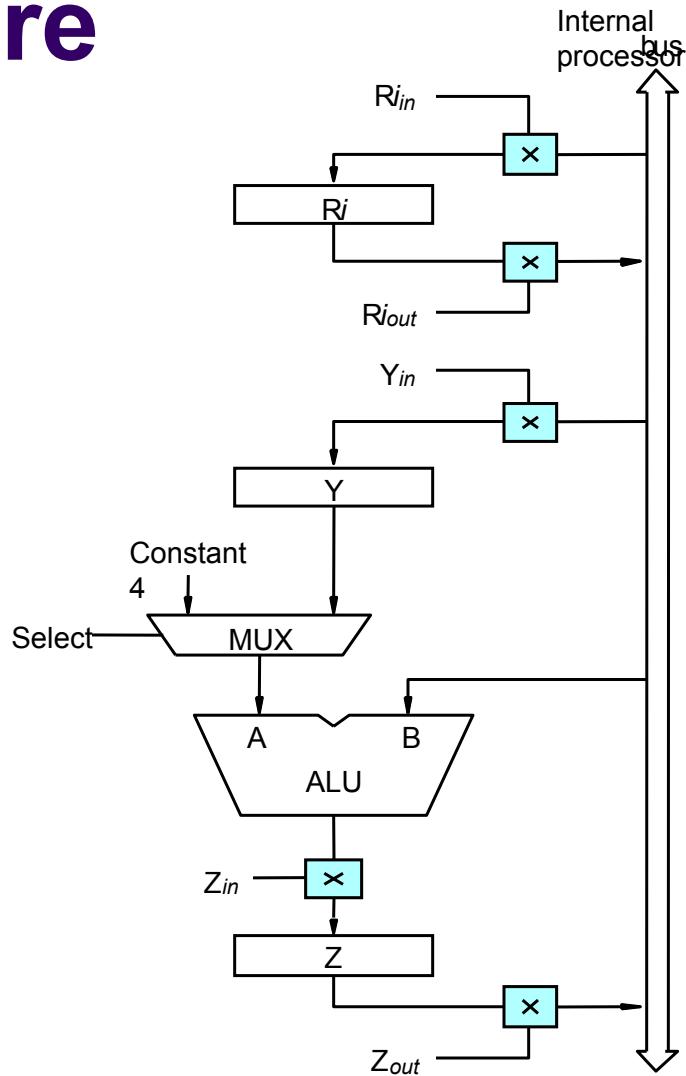


Figure 7.2. Input and output gating for the registers in Figure 7.1.

Execution of a Complete Instruction (for understanding)



Add R1, (R3)

1. PCout , MARin, read, MDRin
2. Add Zin, select4, PCout
3. PCin , Zout, WMFC
4. IRin , MDRout
5. read MARin , R3out
6. Yin, R1out
7. Add Zin, selectY, MDRout
8. R1in , Zout WMFC

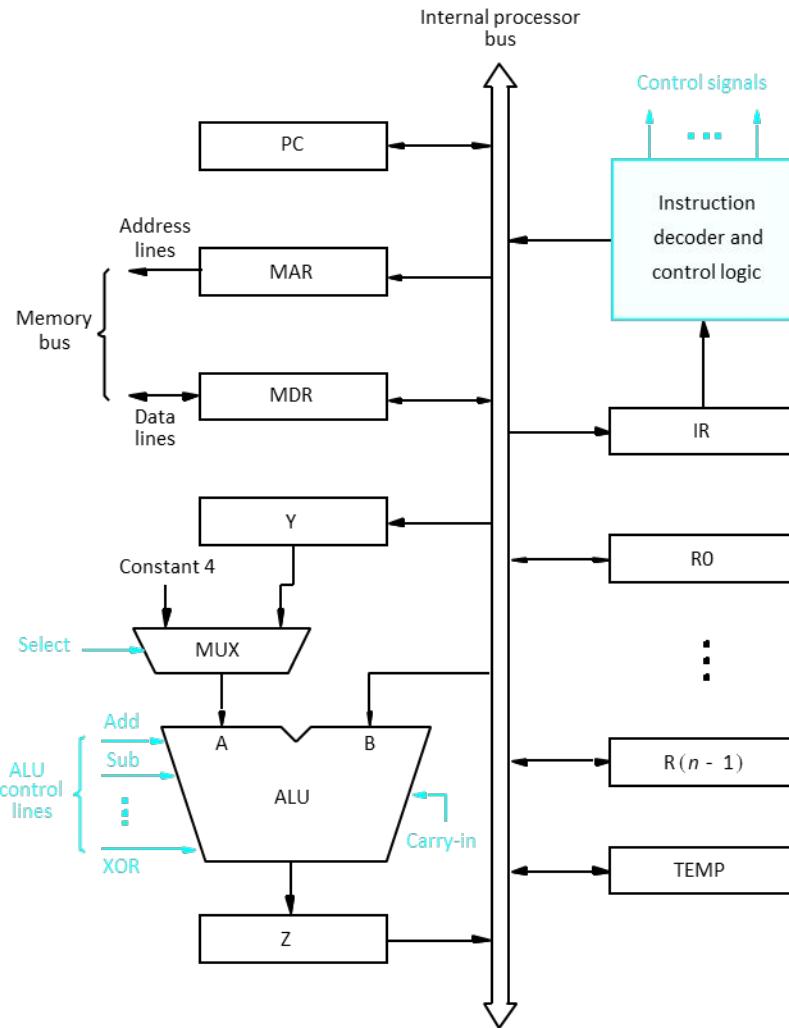
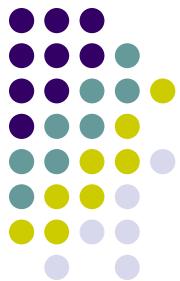


Figure 7.1. Single-bus organization of the datapath inside a processor.

Execution of a Complete Instruction (book)



Add R1, (R3)

1. PCout , MARin, read, Add Zin, select4
2. PCin , Zout, Yin, WMFC
3. IRin , MDRout
4. read MARin , R3out
5. Yin, R1out, WMFC
6. Add Zin, selectY, MDRout
7. R1in , Zout END

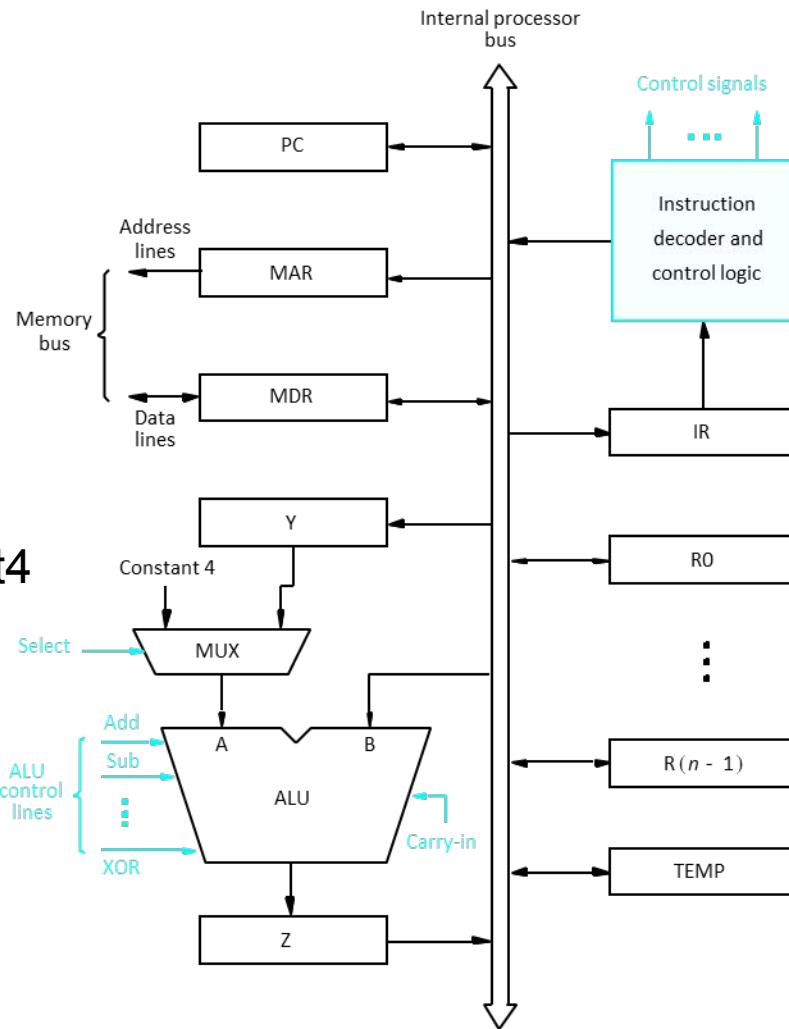
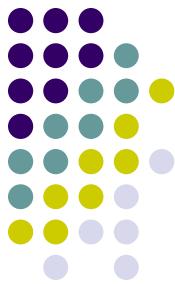


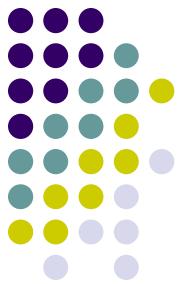
Figure 7.1. Single-bus organization of the datapath inside a processor.

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. PCout , MARin, read, MDRin
 2. Add Zin, select4, PCout
 3. PCin , Zout, WMFC
-
- 4 Add, Z_{in} ,Offset-field-of-IR $_{out}$
 - 5 P $_{in}$, Z $_{out}$, End
C
-

Figure 7.7. Control sequence for an unconditional branch instruction.

Quiz

- What is the control sequence for execution of the instruction
Add R1, R2
including the instruction fetch phase? (Assume single bus architecture)

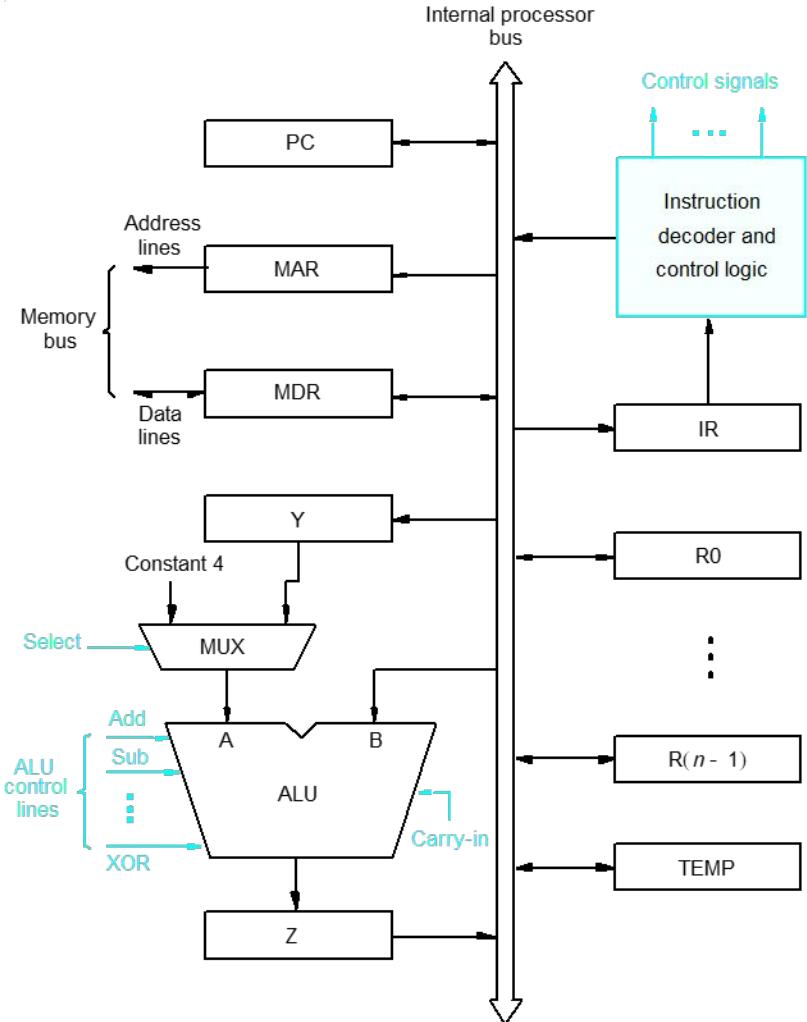
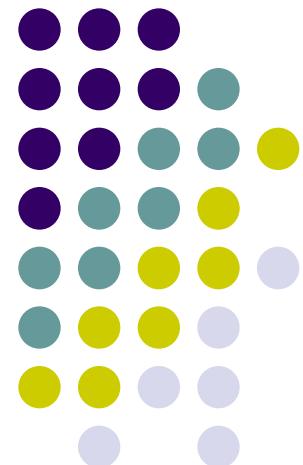
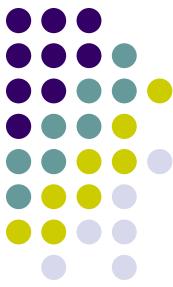


Figure 7.1. Single-bus organization of the datapath inside a proce



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

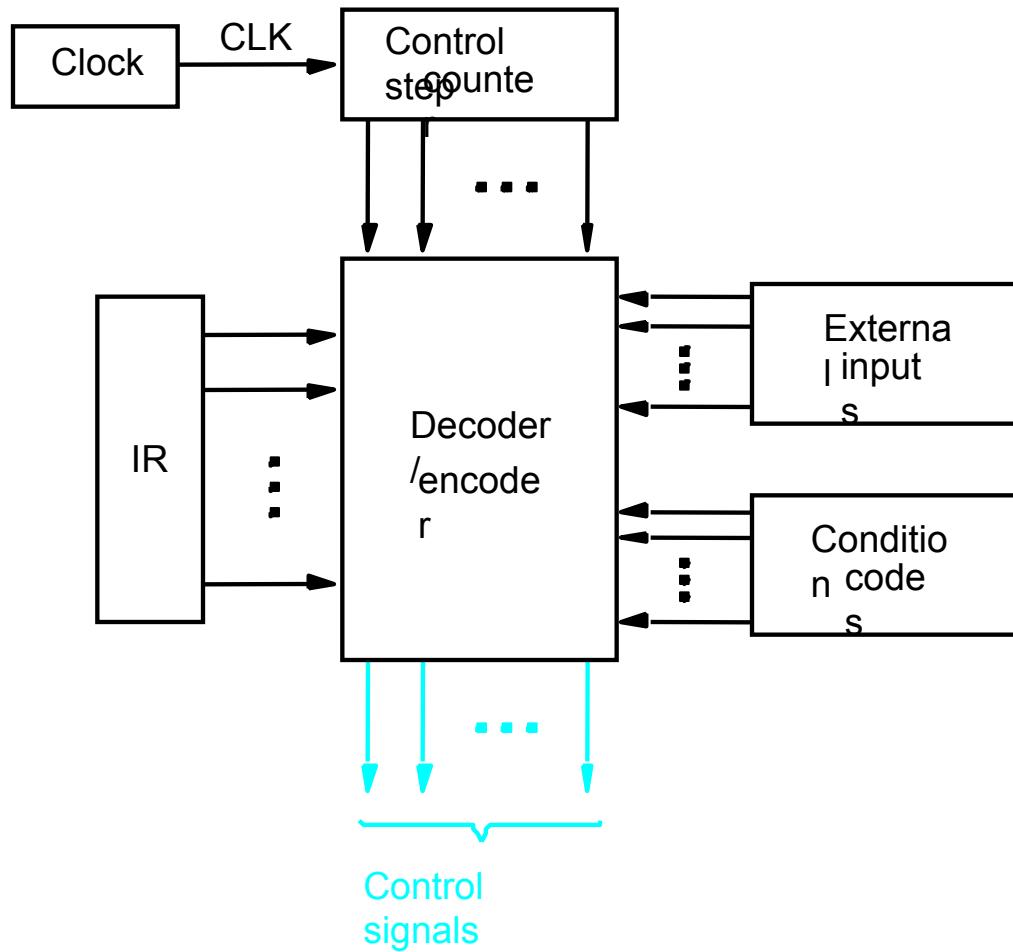


Figure 7.10. Control unit organization.



Detailed Block Description

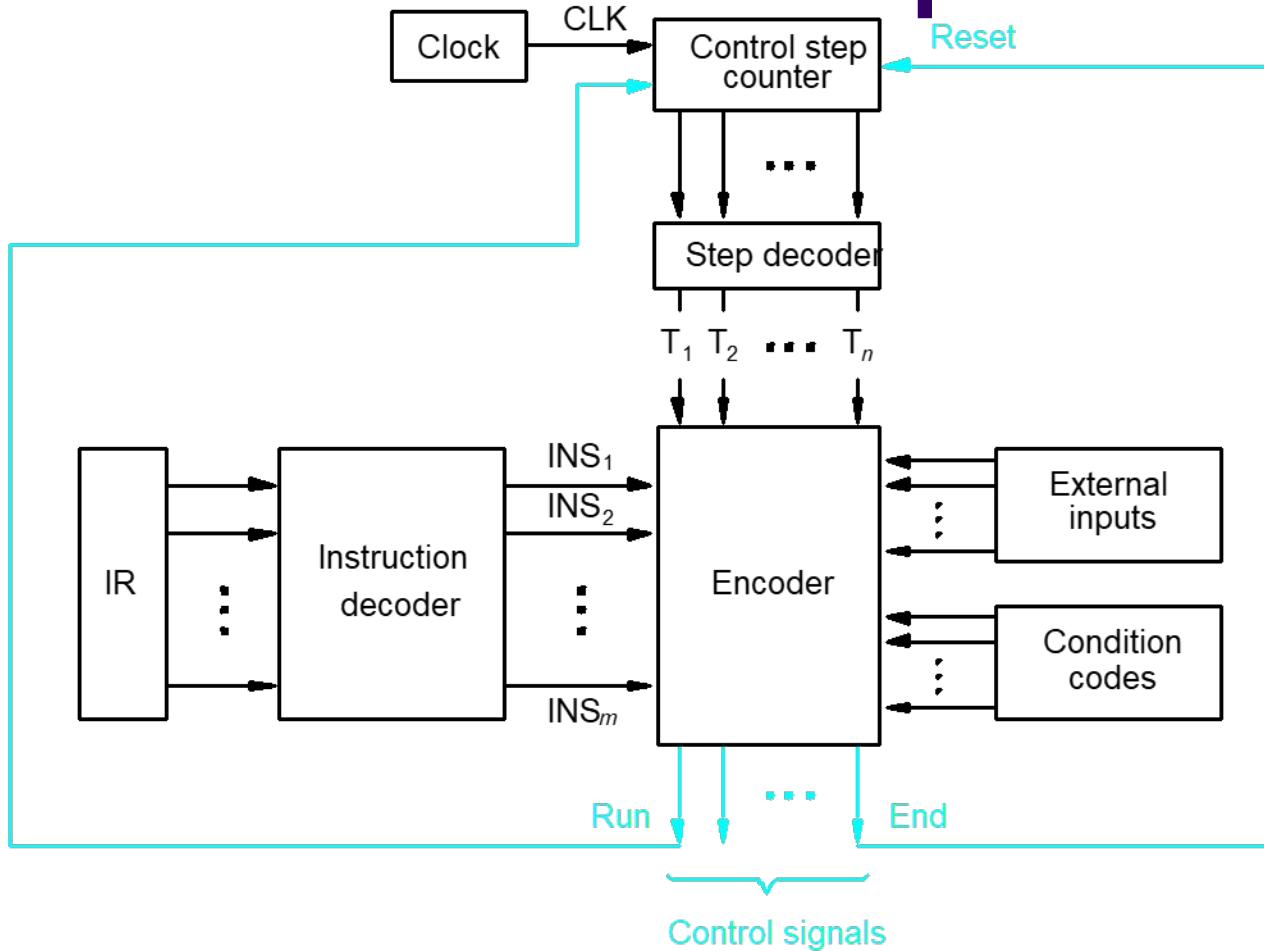


Figure 7.11. Separation of the decoding and encoding function



Generating Z_{in}

- $Z_{in} = T_1 + T_6 \cdot \text{ADD} + T_4 \cdot \text{BR} + \dots$

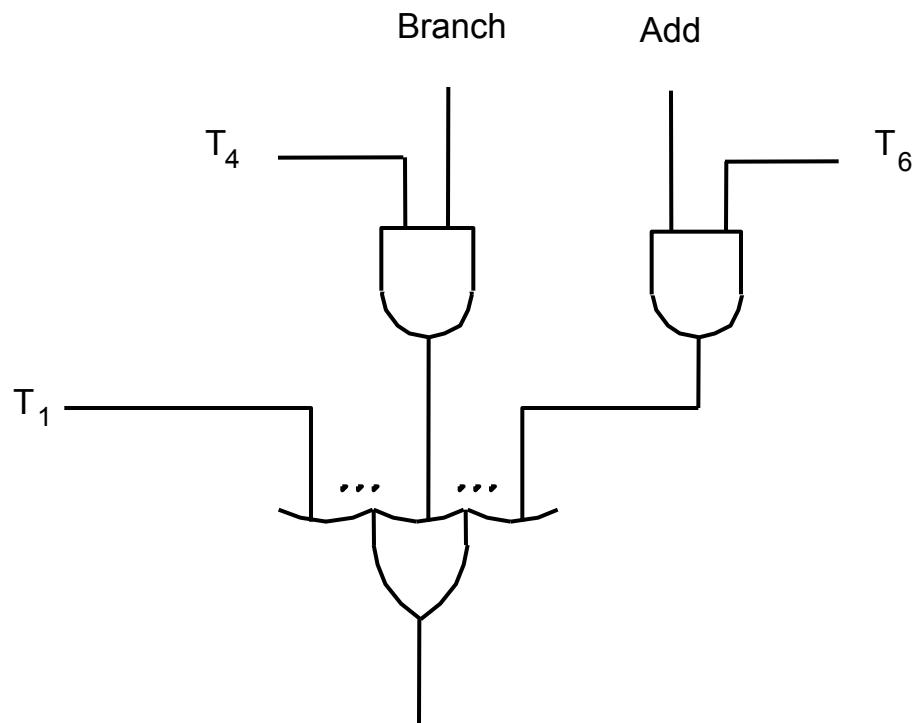


Figure 7.12. Generation of the Z_{in} control signal for the processor in Figure 7.1.



Generating End

- $\text{End} = T_7 \cdot \text{ADD} + T_5 \cdot \text{BR} + (T_5 \cdot N + T_4 \cdot \bar{N}) \cdot \text{BRN} + \dots$

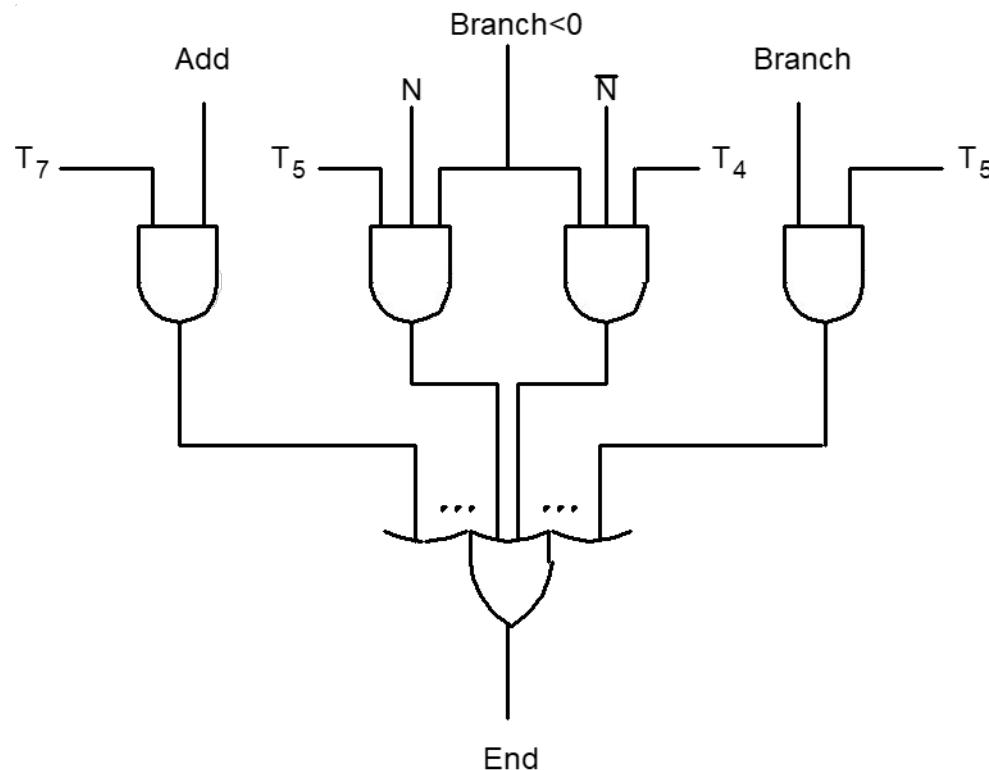
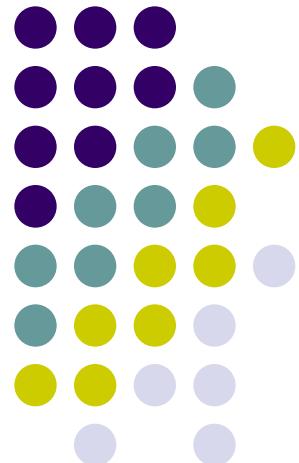
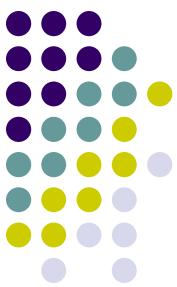


Figure 7.13. Generation of the End control signal.

Microprogrammed Control



Execution of a Complete Instruction



Add R1, (R3)

1. PCout , MARin, read, Add Zin, select4
2. PCin , Zout, Yin, WMFC
3. IRin , MDRout
4. read MARin , R3out
5. Yin, R1out, WMFC
6. Add Zin, selectY, MDRout
7. R1in , Zout END

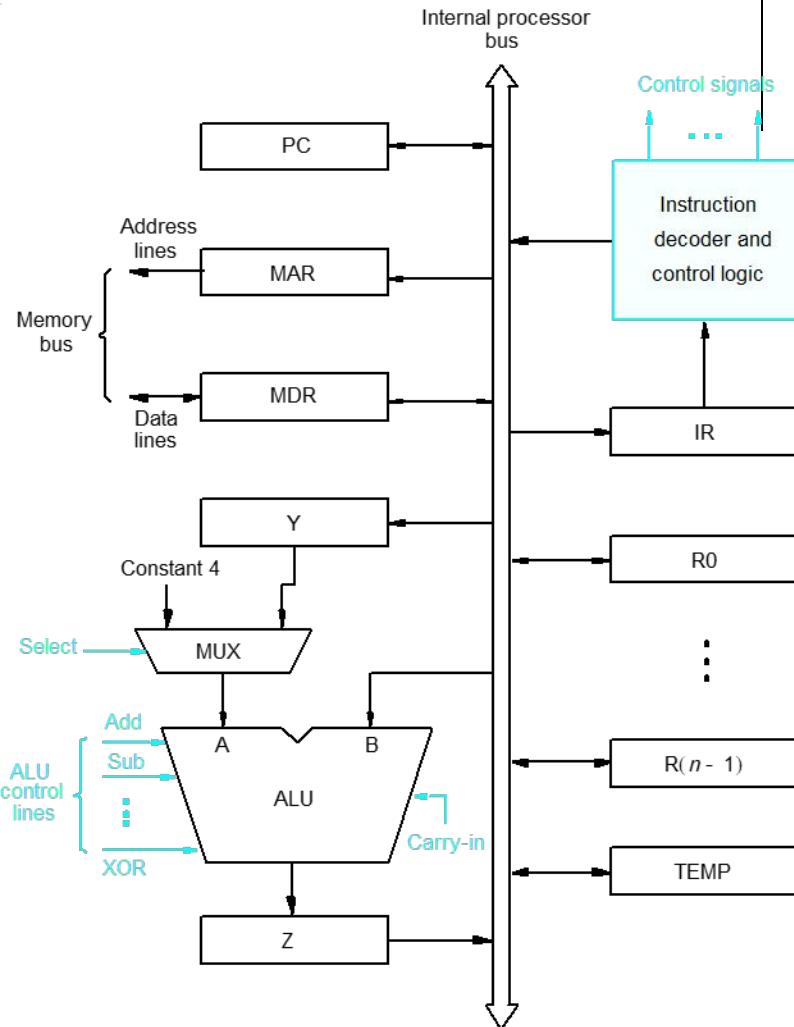


Figure 7.1. Single-bus organization of the datapath inside a proce



Overview

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

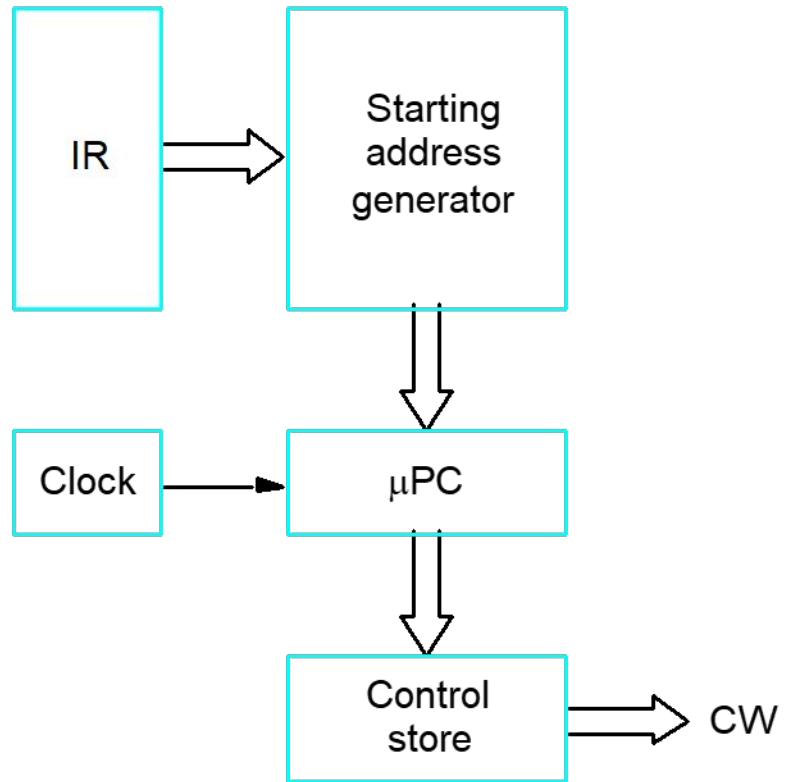
Micro - instruction	..	PC_{in}	PC_{out}	MAR_{in}	Read	MDR_{out}	IR_{in}	Y_{in}	Select	Add	Z_{in}	Z_{out}	$R1_{out}$	$R1_{in}$	$R3_{out}$	W/MFC	End	:
1		0	1	1	1	0	0	0	1	1	1	0	0	0	0	0	0	
2		1	0	0	0	0	0	1	0	0	0	1	0	0	0	1	0	
3		0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	
4		0	0	1	1	0	0	0	0	0	0	0	0	0	1	0	0	
5		0	0	0	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

- Control store



One function cannot be carried out by this simple organization.

Figure 7.16. Basic organization of a microprogrammed control ur



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.

AddressMicroinstruction	
n	
0	PC _{ou} , MAR _i , Read, Select4, Add, Z _i
1	Z _{ou} ^t , PC _{in} , Y _{in} , WM C ⁿ
2	MDR _{ou} , IR _i F
3	Branch t starting address ⁿ appropriate microroutine
..... o f
25	I N=0, then branch t microinstruction0
26	f Offset-field-of-I _{ou} , SelectY, Add, Z _i
27	Z _{ou} , PC _{in} , End ^t n

Figure 7.17. Microroutine for the instruction Branch<0.

Overview

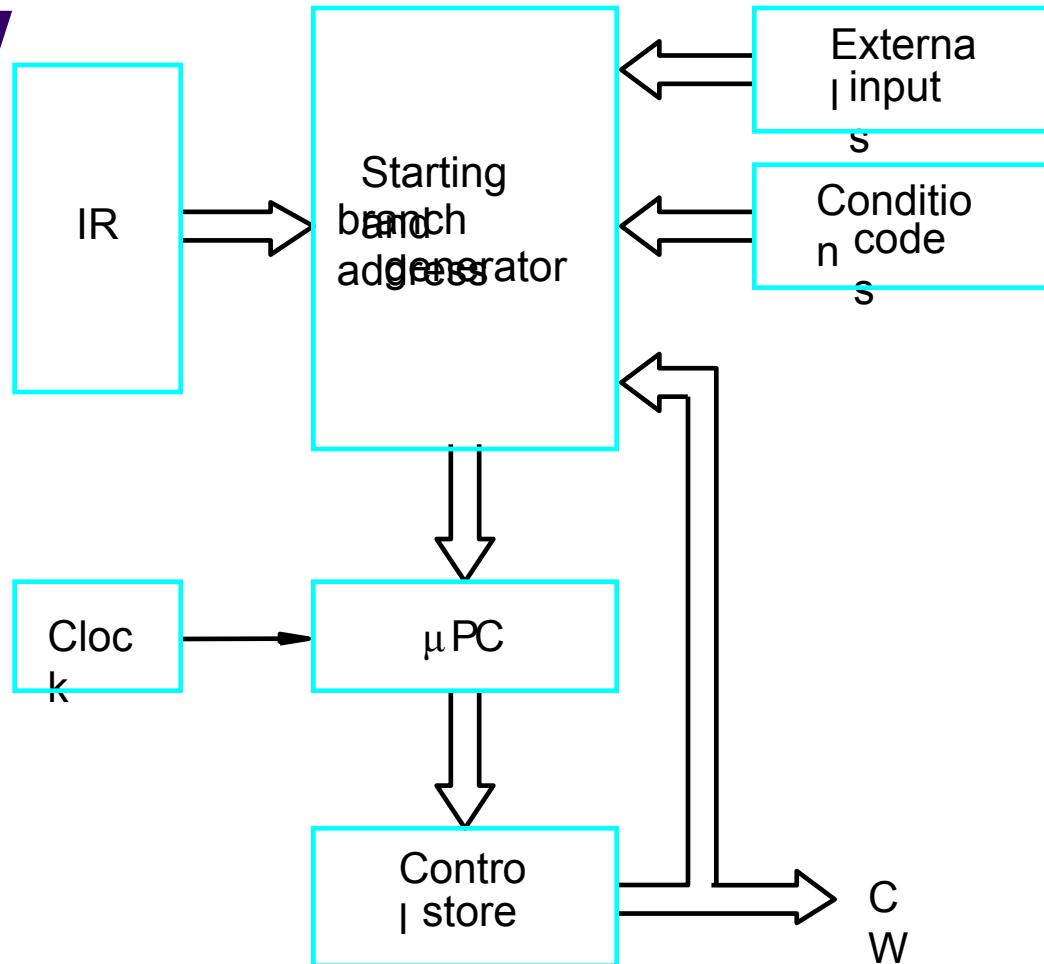
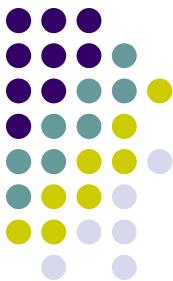


Figure 7.18. Organization of the control unit to allow conditional branching in the microprogram.



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 _n	001: MAR _{in}	0001: Sub	01: Read
0010: MDR _{out}	010: IR _n	010: MDR _n	011: TEMP _{in}	10: Write
0011: Z _{out}	011: Z _n	011: Y _{in}	1111: XOR	
0100: R0 _{out}	100: R0 _n	100: Y _{in}		
0101: R1 _{out}	101: R1 _n			
0110: R2 _{out}	110: R2 _n			
0111: R3 _{out}	111: R3 _n			
1010: TEMP _{out}				
1011: Offset _{out}				

16 ALU
functions

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	

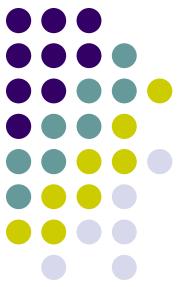
What is the price paid for this scheme?

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



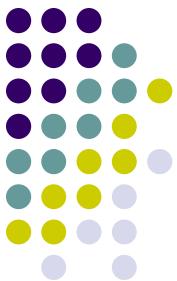
Further Improvement

- 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
- Horizontal organization



Vertical microinstructions

- Width is narrow.
- N control signals are encoded into $\log n$ bits.
- Limited parallelism.



Horizontal microinstructions

- Wide memory word
- High degree of parallel operation possible
- Little encoding of control information



Microprogram Sequencing

- If all microprograms require only straightforward sequential execution of microinstructions except for branches, letting a μ PC 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).



Microinstruction

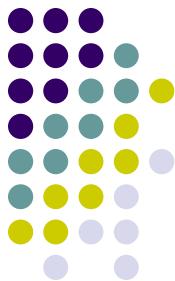
F0	F1	F2	F3
F0 (8 bits)	F1 (3 bits)	F2 (3 bits)	F3 (3 bits)
Address of next microinstruction	000: No transfer 001: PC_{out} 010: MDR_{out} 011: Z_{out} 100: $Rsrc_{out}$ 101: $Rdst_{out}$ 110: $TEMP_{out}$	000: No transfer 001: PC_{in} 010: IR_{in} 011: Z_{in} 100: $Rsrc_{in}$ 101: $Rdst_{in}$	000: No 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	00: No action	0: SelectY	0: No action
0001: Sub	01: Read	1: Select4	1: WMFC
:	10: Write		
1111: XOR			

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

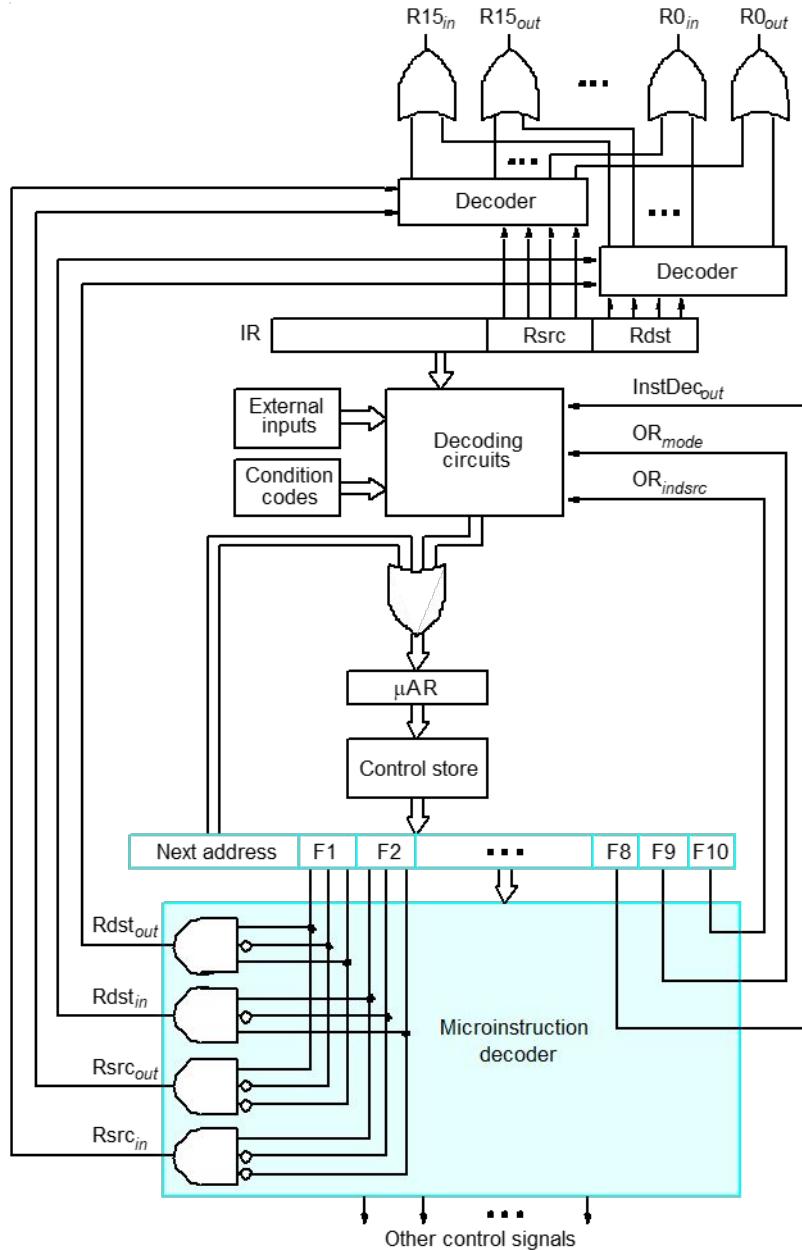


Figure 7.25. Some details of the control-signal-generating circuitry.