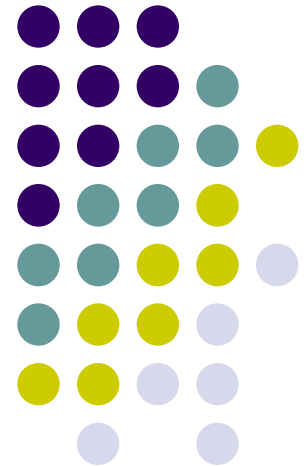
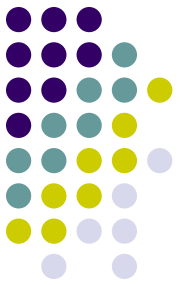


Chapter 7. Basic Processing Unit

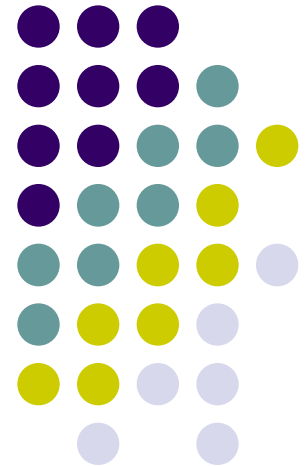


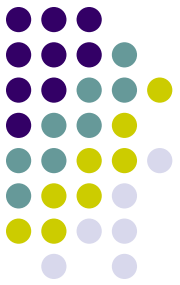


Overview

- 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.

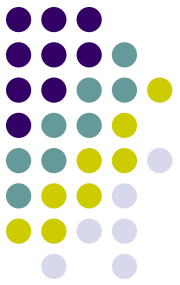
Some Fundamental Concepts





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).

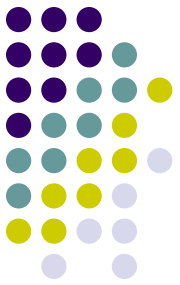
$$IR \leftarrow [[PC]]$$

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

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

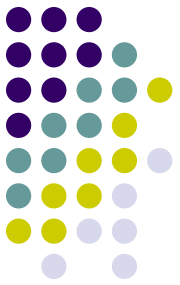
- Carry out the actions specified by the instruction in the IR (execution phase).

Processor Organization



MDR HAS
TWO INPUTS
AND TWO
OUTPUTS

Datapath



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

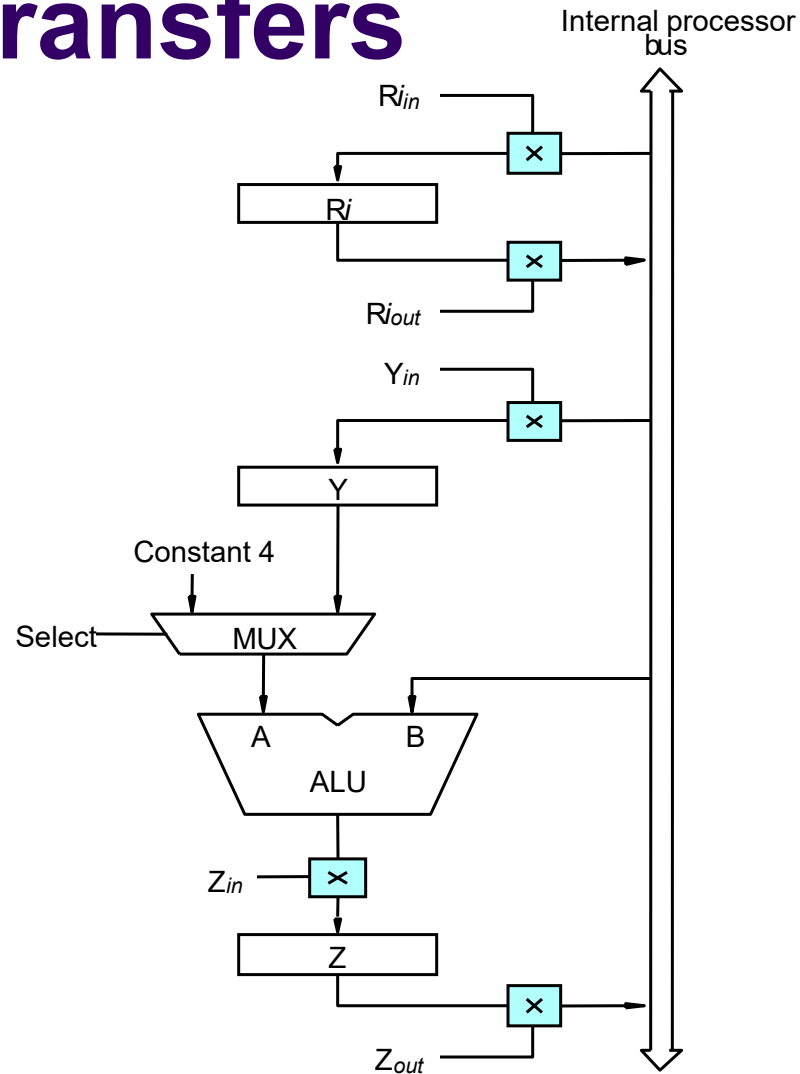
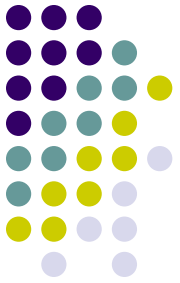
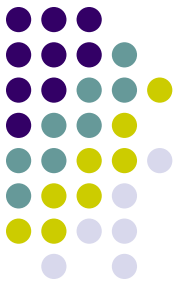


Figure 7.2. Input and output gating for the registers in Figure 7.1.

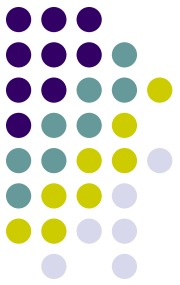
Register Transfers



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

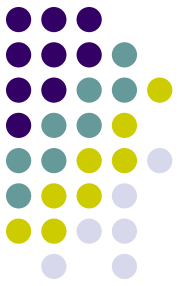
Figure 7.3. Input and output gating for one register bit.

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

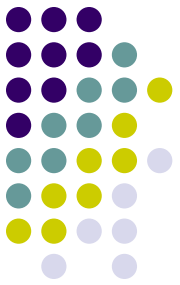
Fetching a Word from Memory



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

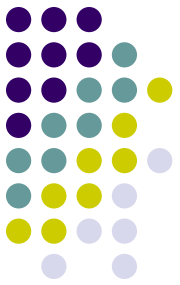
Figure 7.4. Connection and control signals for register MDR.

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

Timing



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

$MAR \leftarrow [R1]$

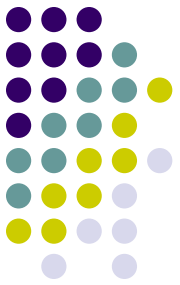
Start a Read operation on the memory bus

Wait for the MFC response from the memory

Load MDR from the memory bus

$R2 \leftarrow [MDR]$

Execution of a Complete Instruction



- Add (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

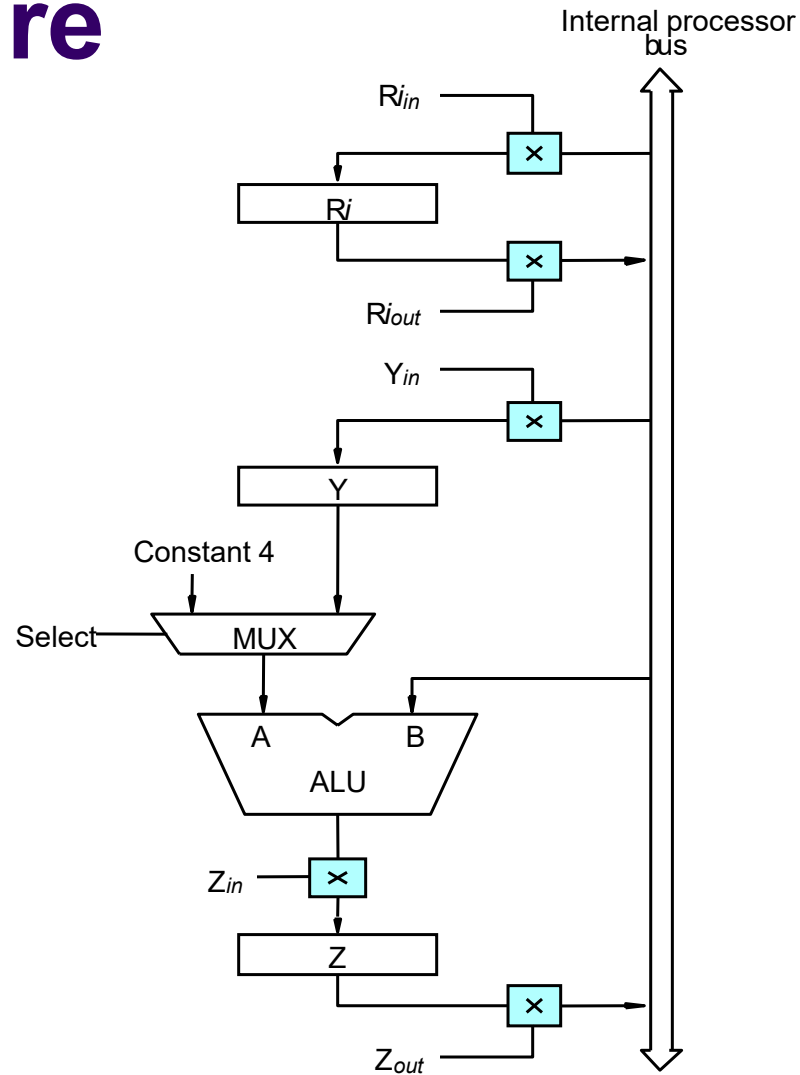
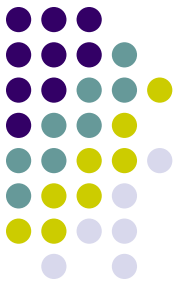
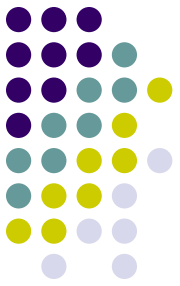


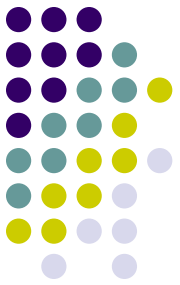
Figure 7.2. Input and output gating for the registers in Figure 7.1.

Execution of a Complete Instruction



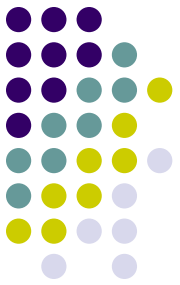
Add (R3), R1

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

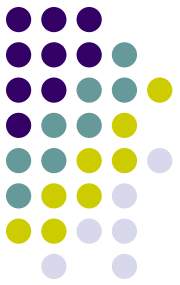


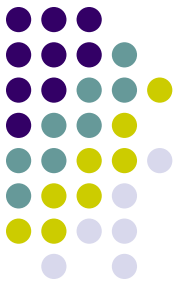
Step Action

- 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





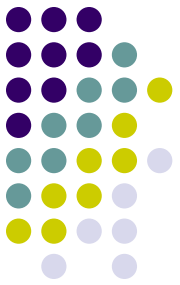
Multiple-Bus Organization

- 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.



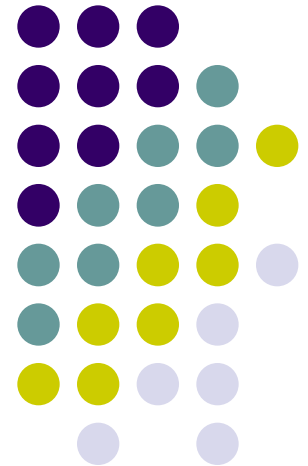
Quiz

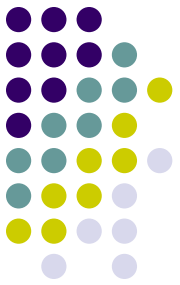
- What is the control sequence for execution of the instruction

Add R1, R2

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

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

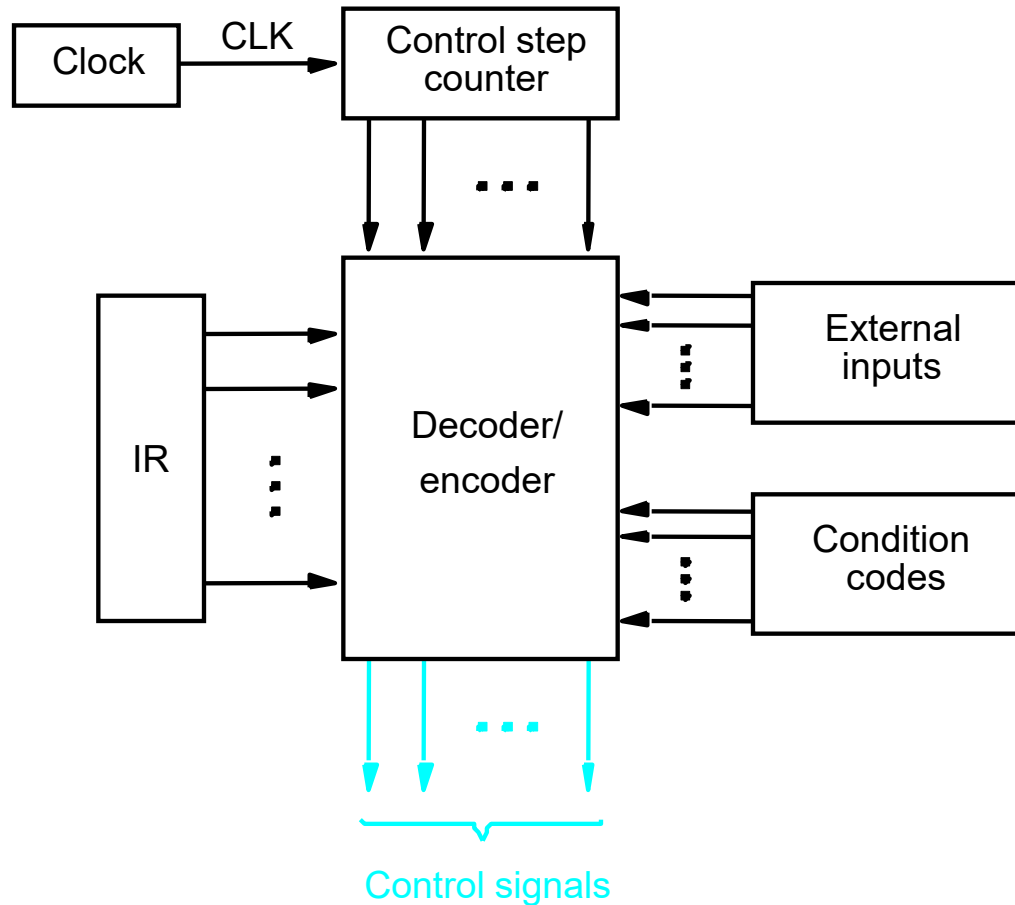
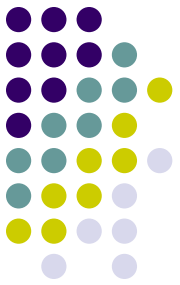
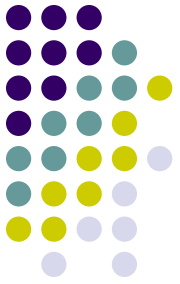
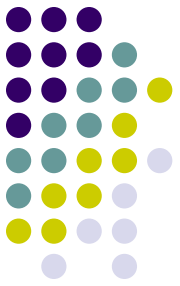


Figure 7.10. Control unit organization.

Detailed Block Description



Generating Z_{in}



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

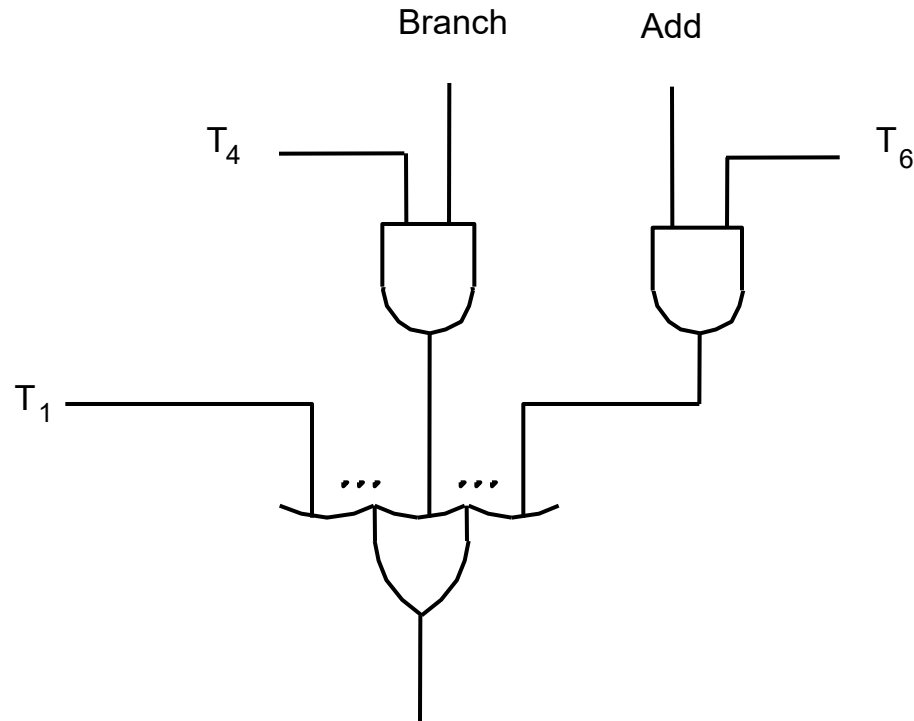
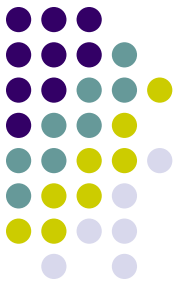


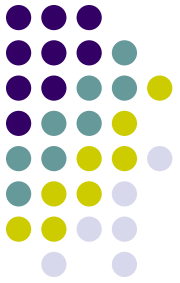
Figure 7.12. Generation of the Z_{in} control signal for the processor in Figure 7.1.

Generating End

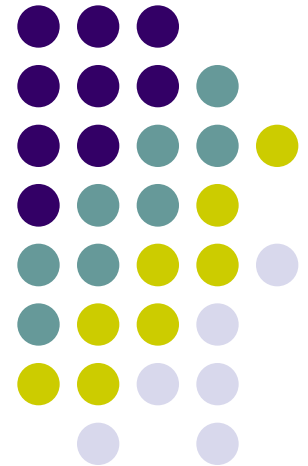


- $\text{End} = T_7 \cdot \text{ADD} + T_5 \cdot \text{BR} + (T_5 \cdot N + T_4 \cdot \overline{N}) \cdot \text{BRN} + \dots$

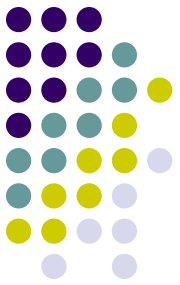
A Complete Processor



Microprogrammed Control

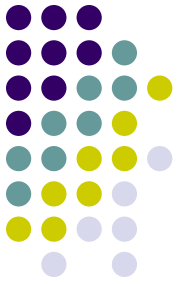


Overview



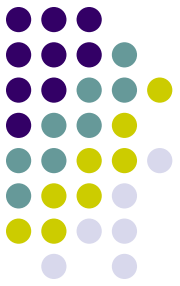
- Control signals are generated by a program similar to machine language programs.
- Control Word (CW); microroutine; microinstruction

Overview

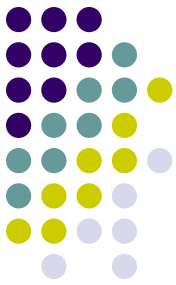


Overview

- Control store



One function
cannot be carried
out by this simple
organization.



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	
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 7.17. Microroutine for the instruction Branch<0.

Overview

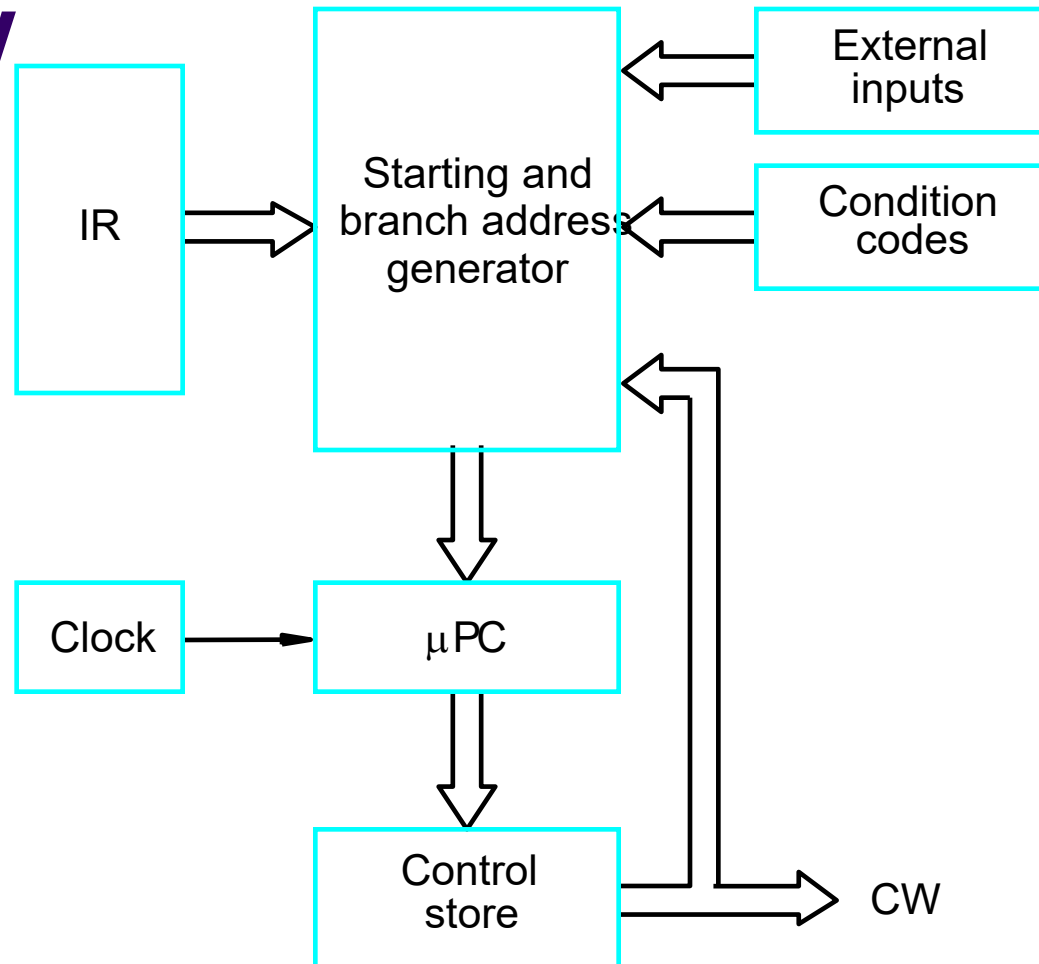
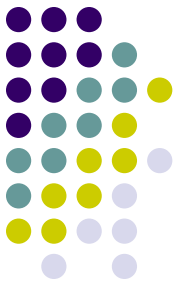


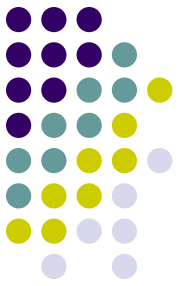
Figure 7.18. Organization of the control unit to allow conditional branching in the microprogram.



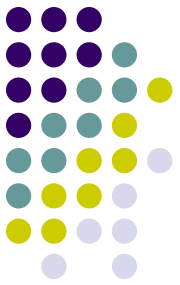
Microinstructions

- 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

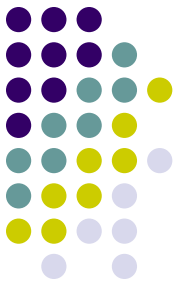


What is the price paid for
this scheme?



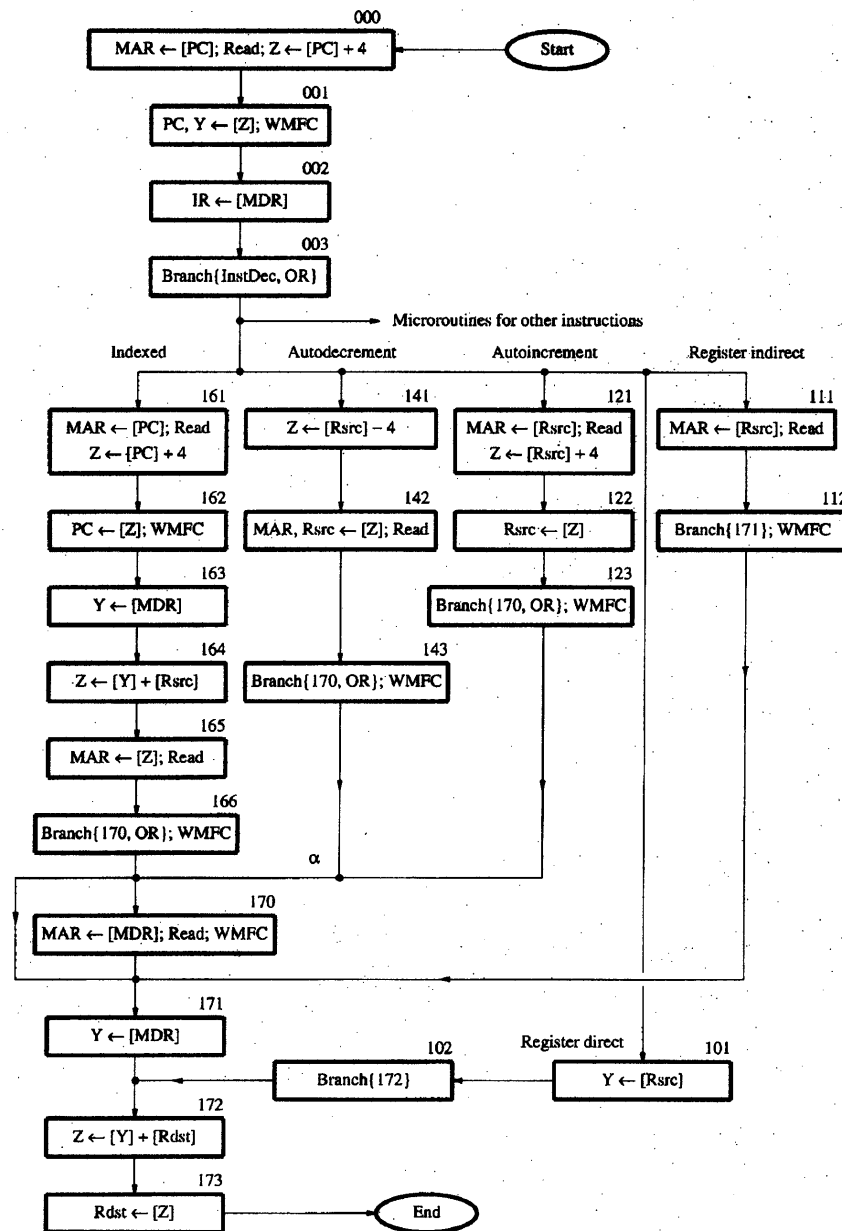
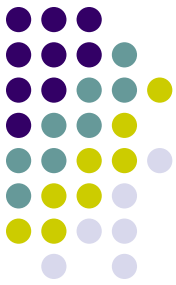
Further Improvement

- 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



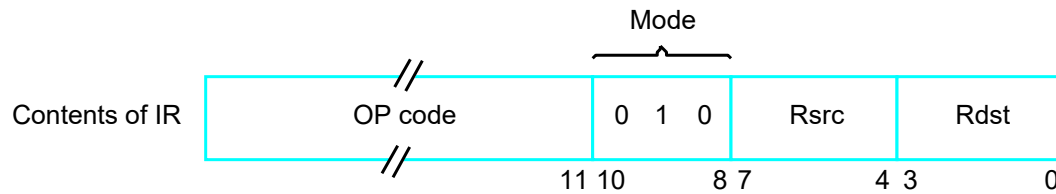
Microprogram Sequencing

- 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).



- Bit-ORing
- Wide-Branch Addressing
- WMFC

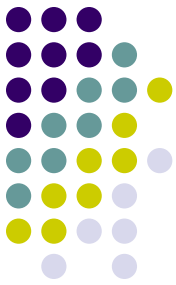
Figure 7.20. Flowchart of a microprogram for the Add src,Rdst instruction.



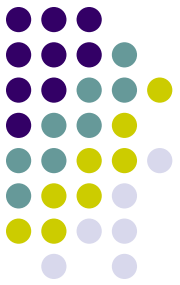
Address (octal)	Microinstruction
000	$PC_{out}, MAR_{in}, \text{Read, Select4, Add, } Z_{in}$
001	$Z_{out}, PC_{in}, Y_{in}, \text{WMFC}$
002	MDR_{out}, IR_{in}
003	$\mu\text{Branch } \{PC \leftarrow 101 \text{ (from Instruction decoder);}$ $\mu PC_{5,4} \leftarrow [IR_{10,9}]; \mu PC_3 \leftarrow [\overline{IR_{10}}]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

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

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

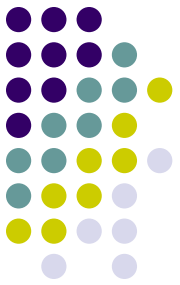


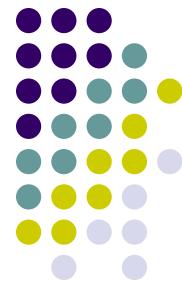
Microinstructions with Next-Address Field



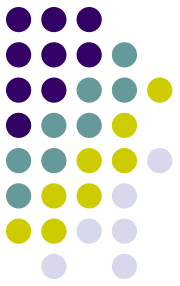
- The microprogram we discussed requires several branch microinstructions, which perform no useful operation in the datapath.
- A powerful alternative approach is to include an address field as a part of every microinstruction to indicate the location of the next microinstruction to be fetched.
- Pros: separate branch microinstructions are virtually eliminated; few limitations in assigning addresses to microinstructions.
- Cons: additional bits for the address field (around 1/6)

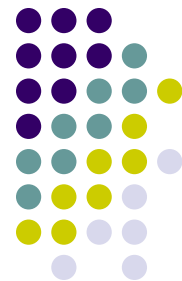
Microinstructions with Next-Address Field





Implementation of the Microroutine





bit-ORing

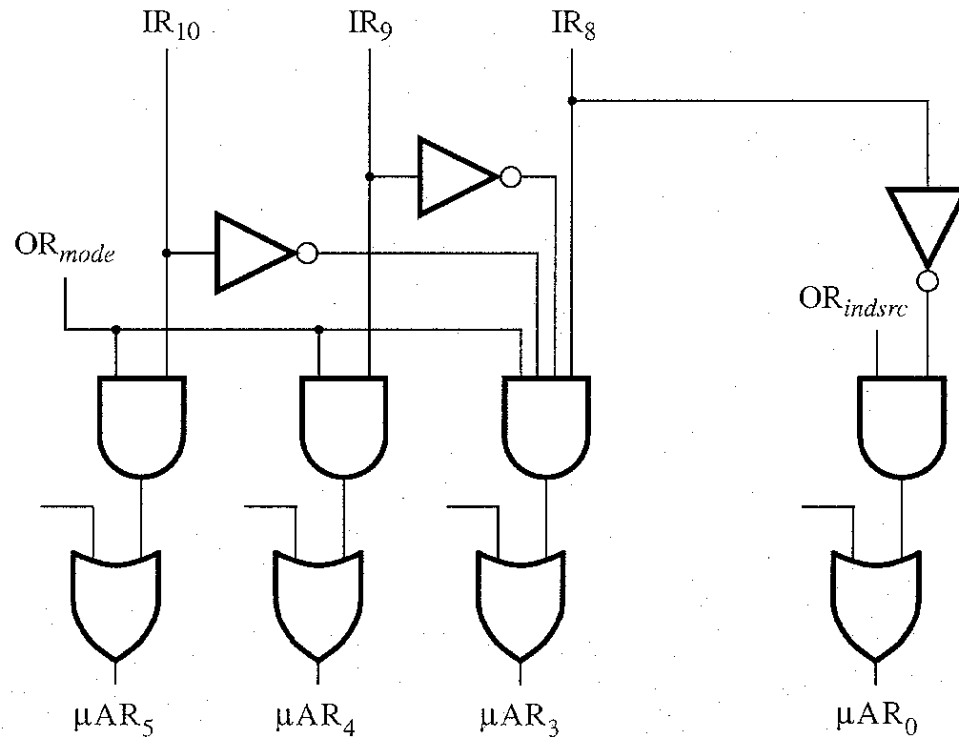
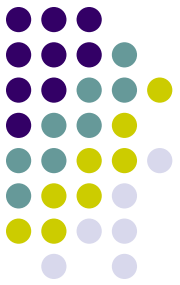


Figure 7.26. Control circuitry for bit-ORing
(part of the decoding circuits in Figure 7.25).

Further Discussions

- Prefetching
- Emulation

