UNIT – IV

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

Datapath

Internal organization of the processor

- ALU
- Registers for temporary storage
- Various digital circuits for executing different micro operations.(gates, MUX,decoders,counters).
- Internal path for movement of data between ALU and registers.
- Driver circuits for transmitting signals to external units.
- Receiver circuits for incoming signals from external units.

- PC:
  - Keeps track of execution of a program
  - Contains the memory address of the next instruction to be fetched and executed.

MAR:

- Holds the address of the location to be accessed.
- I/P of MAR is connected to Internal bus and an O/p to external bus.

MDR:

- Contains data to be written into or read out of the addressed location.
- IT has 2 inputs and 2 Outputs.
- Data can be loaded into MDR either from memory bus or from internal processor bus.

The data and address lines are connected to the internal bus via MDR and MAR

Registers:

- The processor registers R0 to Rn-1 vary considerably from one processor to another.
- Registers are provided for general purpose used by programmer.
- Special purpose registers-index & stack registers.
- Registers Y,Z &TEMP are temporary registers used by processor during the execution of some instruction.

Multiplexer:

- Select either the output of the register Y or a constant value 4 to be provided as input A of the ALU.
- Constant 4 is used by the processor to increment the contents of PC.

ALU:

Used to perform arithmetic and logical operation.

Data Path:

The registers, ALU and interconnecting bus are collectively referred to as the data path.

1.Register Transfers

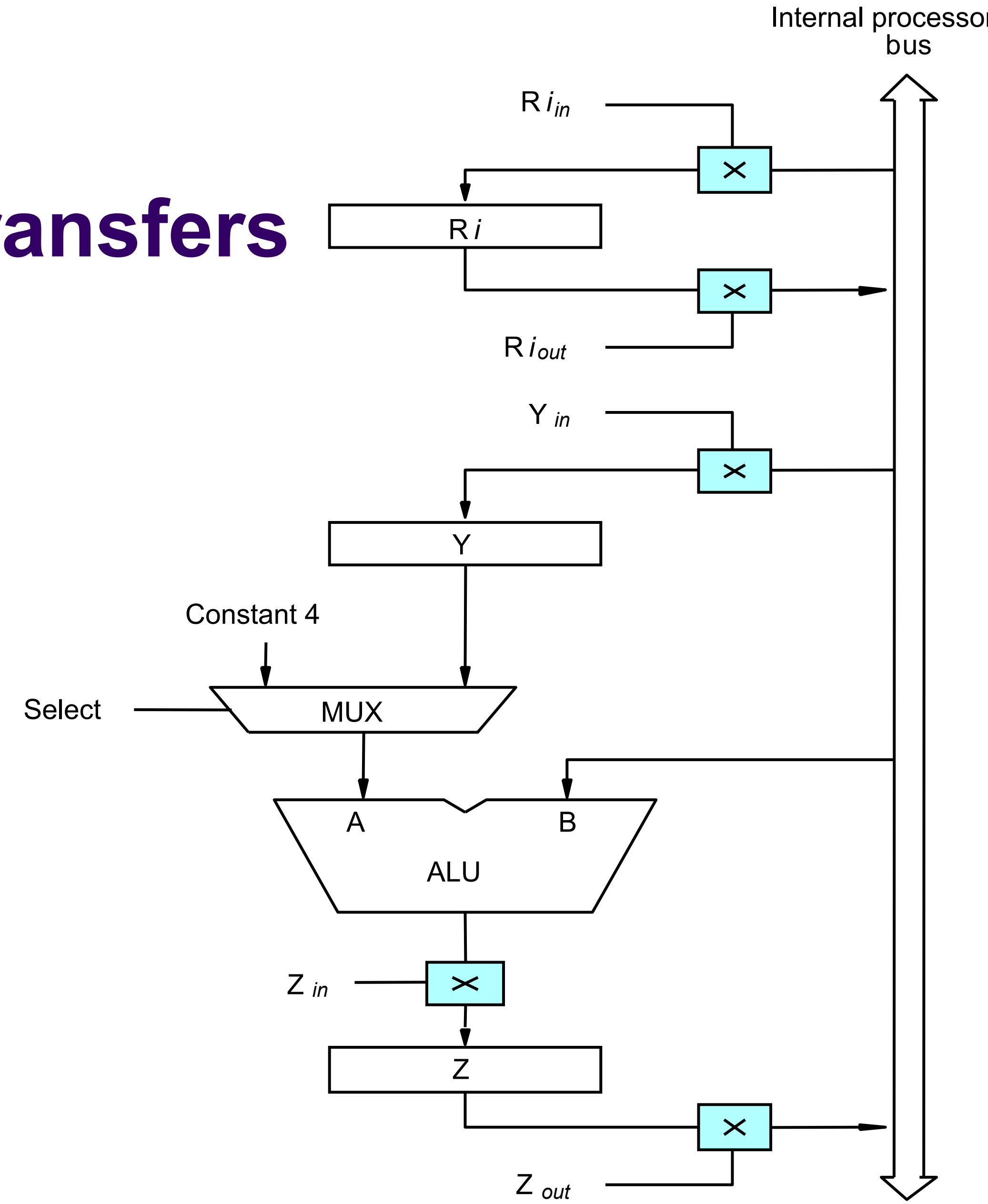

Figure 7.2. Input and output gating for the registers in Figure 7.1.

- The input and output gates for register Ri are controlled by signals isR_{in} and R_{iout} .
- R_{in} Is set to1 – data available on common bus are loaded into Ri.
- R_{iout} Is set to1 – the contents of register are placed on the bus.
- R_{iout} Is set to 0 – the bus can be used for transferring data from other registers .

Data transfer between two registers:

EX:

Transfer the contents of R1 to R4.

1. Enable output of register R1 by setting R1out=1. This places the contents of R1 on the processor bus.
2. Enable input of register R4 by setting R4in=1. This loads the data from the processor bus into register R4.

Architecture

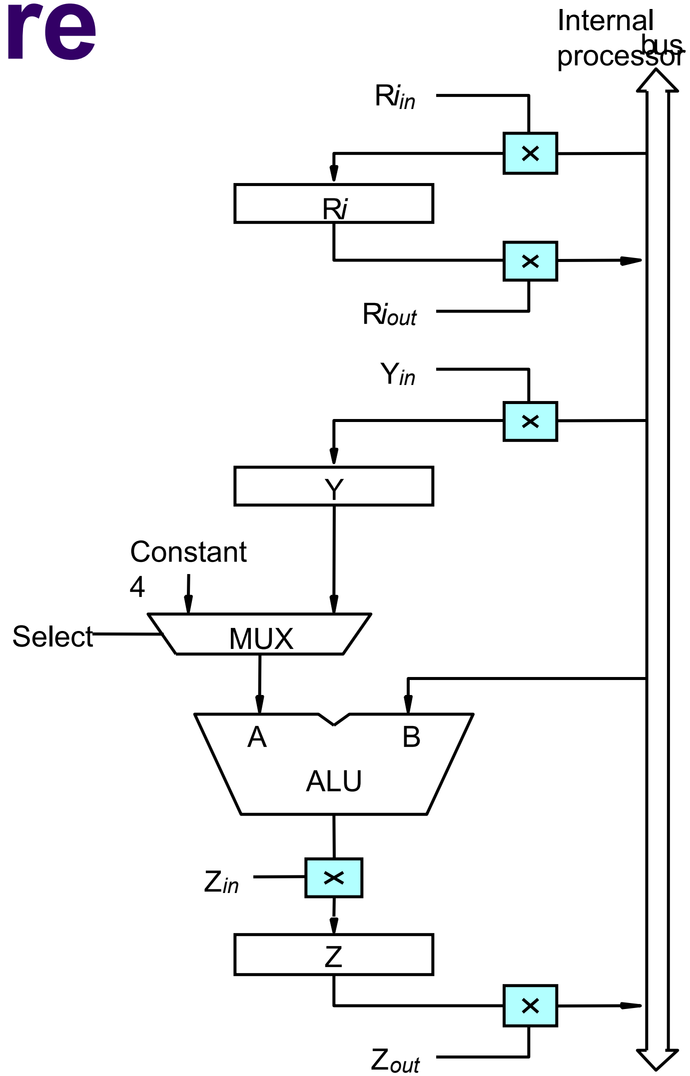

Figure 7.2. Input and output gating for the registers in Figure 7.1.

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

Step 1: Output of the register R1 and input of the register Y are enabled, causing the contents of R1 to be transferred 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.

Step 3: The contents of Z are transferred to the destination register R3.

Register Transfers

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

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

Fetching a Word from Memory

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

Figure 7.4. Connection and control signals for register MDR.

3.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]

Timing

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

- Move (R1), R2

1. R1out, MARin, Read
2. MDRinE, WMFC
3. MDRout, R2in

4.Storing a word in memory

- Address is loaded into MAR
- Data to be written loaded into MDR.
- Write command is issued.
- Example: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

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.
- UnConditional branch

Execution of Branch Instructions

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

- Allow the contents of two different registers to be accessed simultaneously and have their contents placed on buses A and B.
- Allow the data on bus C to be loaded into a third register during the same clock cycle.
- Incrementer unit.
- ALU simply passes one of its two input operands unmodified to bus C

  control signal: R=A or R=B

- General purpose registers are combined into a single block called registers.
- 3 ports,2 output ports –access two different registers and have their contents on buses A and B
- Third port allows data on bus c during same clock cycle.
- Bus A & B are used to transfer the source operands to A & B inputs of the ALU.
- ALU operation is performed.
- The result is transferred to the destination over the bus C.

- ALU may simply pass one of its 2 input operands unmodified to bus C.
- The ALU control signals for such an operation R=A or R=B.
- Incrementer unit is used to increment the PC by 4.
- Using the incrementer eliminates the need to add the constant value 4 to the PC using the main ALU.
- The source for the constant 4 at the ALU input multiplexer can be used to increment other address such as loadmultiple & storemultiple

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.

- Step 1:The contents of PC are passed through the ALU using R=B control signal & loaded into MAR to start a memory read operation

  At the same time PC is incrementer by 4
- Step 2:The processor waits for MFC
- Step 3: Loads the data ,received into MDR ,then transfers them to IR.
- Step 4: The execution phase of the instruction requires only one control step to complete.

Exercise

- 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

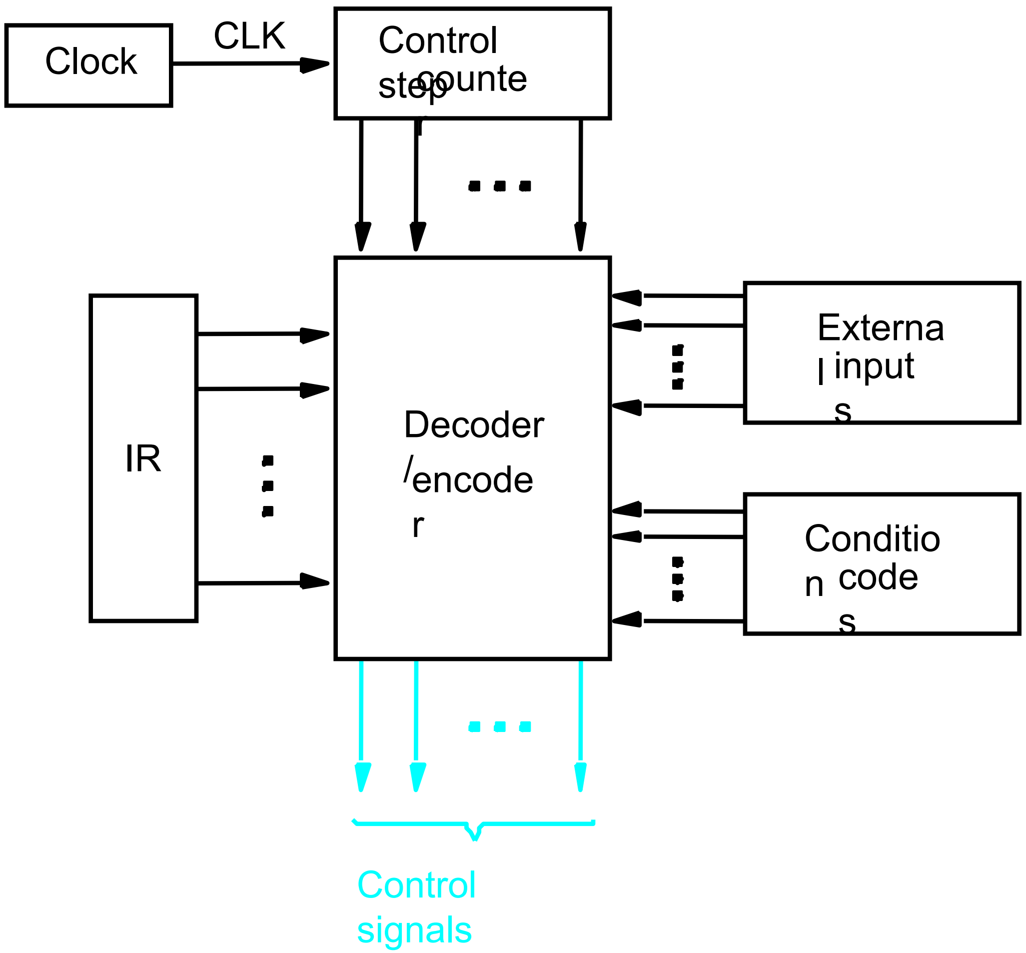

Figure 7.10. Control unit organization.

Detailed Block Description

Generating Z_{in}

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

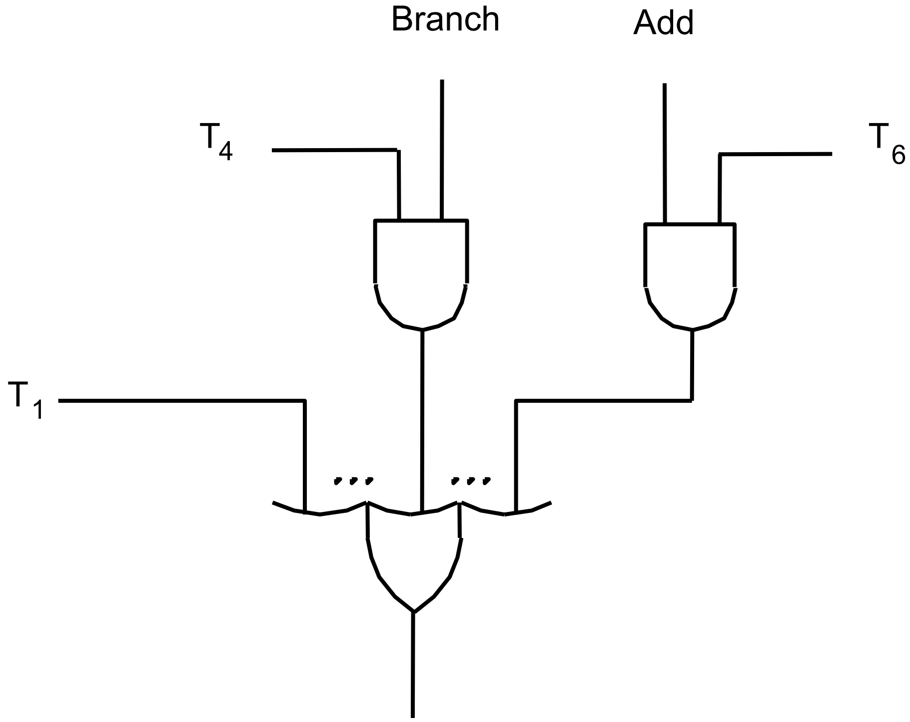

Figure 7.12. Generation of the Z_{in} control signal for the processor in Figure 7.1.

Generating End

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

A Complete Processor

Microprogrammed Control

Microprogrammed Control

- Control signals are generated by a program similar to machine language programs.
- Control Word (CW); microroutine; microinstruction : Textbook page430

Overview

Textbook page 421

Basic organization of a microprogrammed control unit

- Control store

One function cannot be carried out by this simple organization.

Conditional branch

- 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 7.17. Microroutine for the instruction Branch<0.

Microprogrammed Control

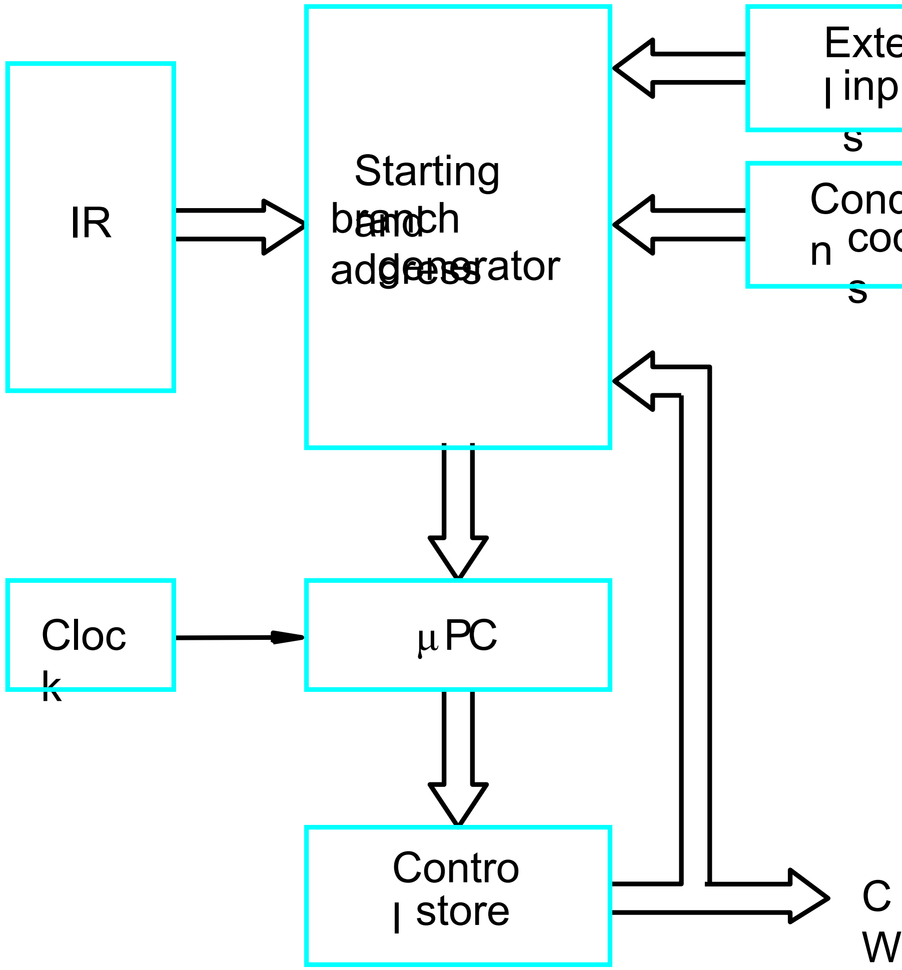

Figure 7.18. Organization of the control unit to allow conditional branching in the microprogram.

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?

Require a little more hardware

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

Textbook page 434

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).

Textbook page 436

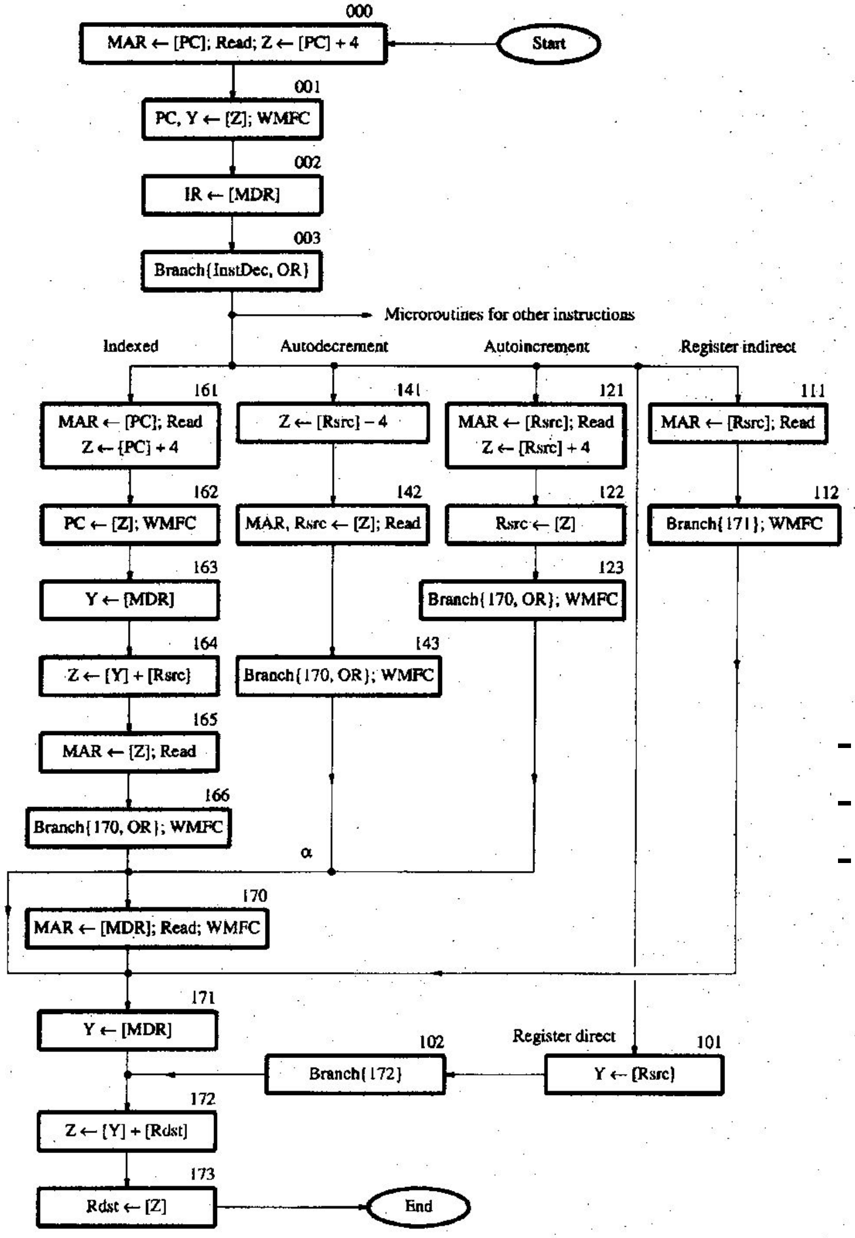

Figure 7.20. Flowchart of a microprogram for the Add src,Rdst instruction.

- Bit-ORing
- Wide-Branch Addressing
- WMFC

Textbook page 439

Contents of IR

OP code	Mode: 0 1 0	Rsrc	Rdst
11	10 … 8	7 … 4	3 … 0

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.

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

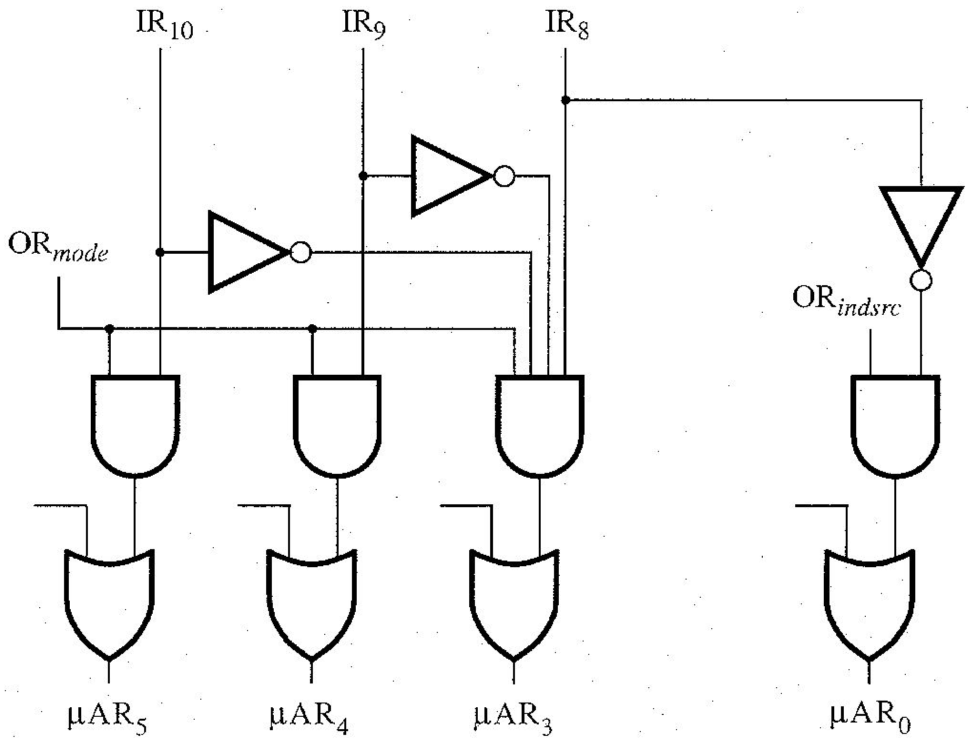

Figure 7.26. Control circuitry for bit-ORing

(part of the decoding circuits in Figure 7.25).

Refrences

- Computer Organization By Carl Hamacher, Zvonko Vranesic, Safwat Zaky, fifth Edition, McGraw-Hill, ISBN 007-120411-3