UNIT 7: BASIC PROCESSING UNIT

SOME FUNDAMENTAL CONCEPTS

- To execute an instruction, processor has to perform following 3 steps:
  1) Fetch contents of memory-location pointed to by PC. Content of this location is an instruction to be executed. The instructions are loaded into IR, Symbolically, this operation can be written as
     IR← [[PC]]
  2) Increment PC by 4
     PC← [PC] +4
  3) Carry out the actions specified by instruction (in the IR).
- The first 2 steps are referred to as fetch phase;
  Step 3 is referred to as execution phase.

SINGLE BUS ORGANIZATION

- MDR has 2 inputs and 2 outputs. Data may be loaded
  - → into MDR either from memory-bus (external) or
  - → from processor-bus (internal).
- MAR"s input is connected to internal-bus, and
  MAR"s output is connected to external-bus.
- Instruction-decoder & control-unit is responsible for
  - → issuing the signals that control the operation of all the units inside the processor (and for interacting with memory bus).
  - → implementing the actions specified by the instruction (loaded in the IR)
- Registers R0 through R(n-1) are provided for general purpose use by programmer.
- Three registers Y, Z & TEMP are used by processor for temporary storage during execution of some instructions. These are transparent to the programmer i.e. programmer need not be concerned with them because they are never referenced explicitly by any instruction.
- MUX(Multiplexer) selects either
  - → output of Y or
  - → constant-value 4(is used to increment PC content).This is provided as input A of ALU.
- B input of ALU is obtained directly from processor-bus.
- As instruction execution progresses, data are transferred from one register to another, often passing through ALU to perform arithmetic or logic operation.
- An instruction can be executed by performing one or more of the following operations:
  1) Transfer a word of data from one processor-register to another or to the ALU.
  2) Perform arithmetic or a logic operation and store the result in a processor-register.
  3) Fetch the contents of a given memory-location and load them into a processor-register.
  4) Store a word of data from a processor-register into a given memory-location.

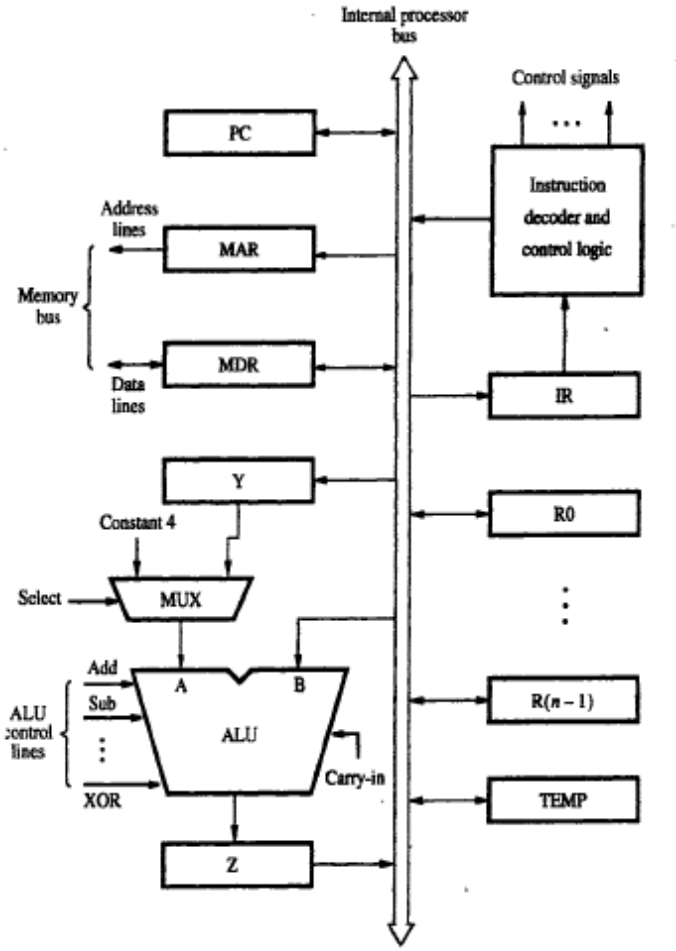


Figure 7.1 Single-bus organization of the datapath inside a processor.

REGISTER TRANSFERS

• Instruction execution involves a sequence of steps in which data are transferred from one register to another.

• Input & output of register Ri is connected to bus via switches controlled by 2 control-signals: Ri_{in} & Ri_{out}. These are called *gating signals.*

• When Ri_{in}=1, data on bus is loaded into Ri.

Similarly, when Ri_{out}=1, content of Ri is placed on bus.

• When Ri_{out}=0, bus can be used for transferring data from other registers.

• All operations and data transfers within the processor take place within time-periods defined by the processor-clock.

• When edge-triggered flip-flops are not used, 2 or more clock-signals may be needed to guarantee proper transfer of data. This is known as *multiphase clocking*.

Input & Output Gating for one Register Bit

• A 2-input multiplexer is used to select the data applied to the input of an edge-triggered D flip-flop.

• When Ri_{in}=1, mux selects data on bus. This data will be loaded into flip-flop at rising-edge of clock.

When Ri_{in}=0, mux feeds back the value currently stored in flip-flop.

• Q output of flip-flop is connected to bus via a tri-state gate.

When Ri_{out}=0, gate's output is in the high-impedance state. (This corresponds to the open-circuit state of a switch).

When Ri_{out}=1, the gate drives the bus to 0 or 1, depending on the value of Q.

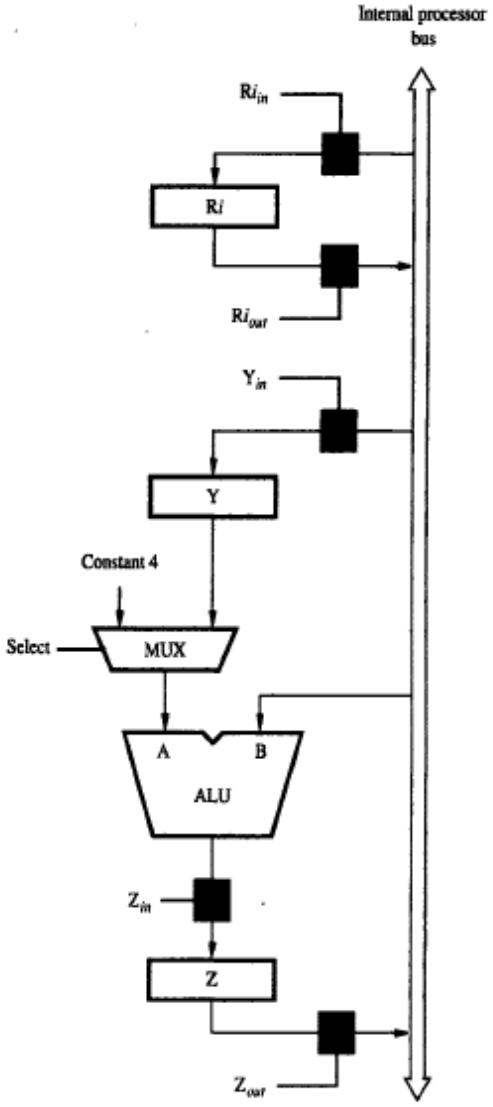


Figure 7.2 Input and output gating for the registers in Figure 7.1.

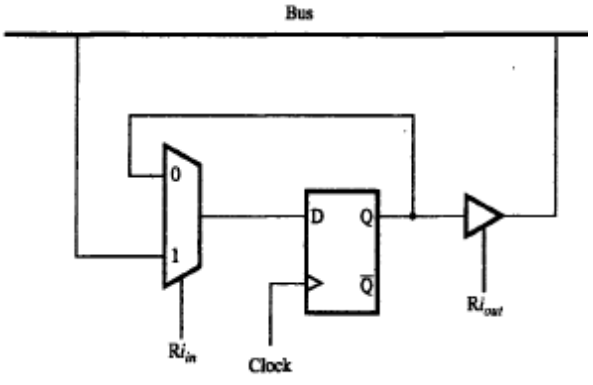


Figure 7.3 Input and output gating for one register bit.

PERFORMING AN ARITHMETIC OR LOGIC OPERATION

- The ALU performs arithmetic operations on the 2 operands applied to its A and B inputs.
- One of the operands is output of MUX & the other operand is obtained directly from bus.
- The result (produced by the ALU) is stored temporarily in register Z.
- The sequence of operations for [R3]←[R1]+[R2] is as follows
  1) $R1_{out}$, Y_{in} //transfer the contents of R1 to Y register
  2) $R2_{out}$, SelectY, Add, Z_{in} //R2 contents are transferred directly to B input of ALU. // The numbers of added. Sum stored in register Z
  3) Z_{out}, $R3_{in}$ //sum is transferred to register R3
- The signals are activated for the duration of the clock cycle corresponding to that step. All other signals are inactive.

Write the complete control sequence for the instruction : Move (R_s),R_d

- This instruction copies the contents of memory-location pointed to by R_s into R_d. This is a memory read operation. This requires the following actions
  → fetch the instruction
  → fetch the operand (i.e. the contents of the memory-location pointed by R_s).
  → transfer the data to R_d.
- The control-sequence is written as follows
  1) PC_{out}, MAR_{in}, Read, Select4, Add, Z_{in}
  2) Z_{out}, PC_{in}, Y_{in}, WMFC
  3) MDR_{out}, IR_{in}
  4) R_s, MAR_{in}, Read
  5) MDR_{inE}, WMFC
  6) MDR_{out}, R_d, End

FETCHING A WORD FROM MEMORY

• To fetch instruction/data from memory, processor transfers required address to MAR (whose output is connected to address-lines of memory-bus).

At the same time, processor issues Read signal on control-lines of memory-bus.

• When requested-data are received from memory, they are stored in MDR. From MDR, they are transferred to other registers

• MFC (Memory Function Completed): Addressed-device sets MFC to 1 to indicate that the contents of the specified location

→ have been read &

→ are available on data-lines of memory-bus

• Consider the instruction Move (R1),R2. The sequence of steps is:

1) $R1_{out}$, MAR_{in}, Read ;desired address is loaded into MAR & Read command is issued
2) MDR_{inE}, WMFC ;load MDR from memory bus & Wait for MFC response from memory
3) MDR_{out}, $R2_{in}$;load R2 from MDR

where WMFC=control signal that causes processor's control circuitry to wait for arrival of MFC signal

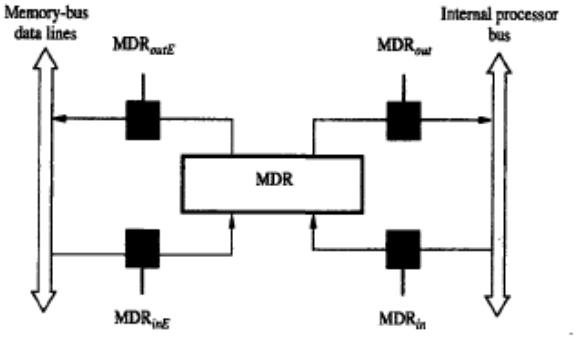


Figure 7.4 Connection and control signals for register MDR.

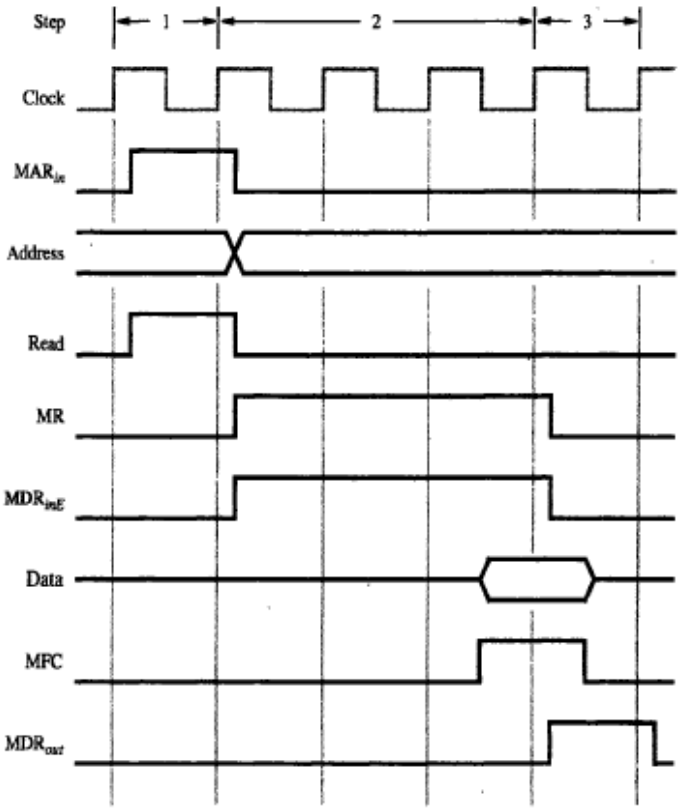


Figure 7.5 Timing of a memory Read operation.

Storing a Word in Memory

• Consider the instruction *Move R2,(R1).* This requires the following sequence:

1) $R1_{out}$, MAR_{in} ;desired address is loaded into MAR
2) $R2_{out}$, MDR_{in}, Write ;data to be written are loaded into MDR & Write command is issued
3) MDR_{outE}, WMFC ;load data into memory location pointed by R1 from MDR

EXECUTION OF A COMPLETE INSTRUCTION

• Consider the instruction *Add (R3),R1* which adds the contents of a memory-location pointed by R3 to register R1. Executing this instruction requires the following actions:

1) Fetch the instruction.
2) Fetch the first operand.
3) Perform the addition.
4) Load the result into R1.

• Control sequence for execution of this instruction is as follows

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

• Instruction execution proceeds as follows:

Step1--> The instruction-fetch operation is initiated by loading contents of PC into MAR & sending a Read request to memory. The Select signal is set to Select4, which causes the Mux to select constant 4. This value is added to operand at input B (PC"s content), and the result is stored in Z
Step2--> Updated value in Z is moved to PC.
Step3--> Fetched instruction is moved into MDR and then to IR.
Step4--> Contents of R3 are loaded into MAR & a memory read signal is issued.
Step5--> Contents of R1 are transferred to Y to prepare for addition.
Step6--> When Read operation is completed, memory-operand is available in MDR, and the addition is performed.
Step7--> Sum is stored in Z, then transferred to R1.The End signal causes a new instruction fetch cycle to begin by returning to step1.

BRANCHING INSTRUCTIONS

• Control sequence for an unconditional branch instruction is as follows:

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

• The processing starts, as usual, the fetch phase ends in step3.
• In step 4, the offset-value is extracted from IR by instruction-decoding circuit.
• Since the updated value of PC is already available in register Y, the offset X is gated onto the bus, and an addition operation is performed.
• In step 5, the result, which is the branch-address, is loaded into the PC.
• The offset X used in a branch instruction is usually the difference between the branch target-address and the address immediately following the branch instruction. (For example, if the branch instruction is at location 1000 and branch target-address is 1200, then the value of X must be 196, since the PC will be containing the address 1004 after fetching the instruction at location 1000).
• In case of conditional branch, we need to check the status of the condition-codes before loading a new value into the PC.

e.g.: Offset-field-of-IR_{out}, Add, Z_{in}, If N=0 then End
If N=0, processor returns to step 1 immediately after step 4.
If N=1, step 5 is performed to load a new value into PC.

MULTIPLE BUS ORGANIZATION

- All general-purpose registers are combined into a single block called the *register file*.
- Register-file has 3 ports. There are 2 outputs allowing the contents of 2 different registers to be simultaneously placed on the buses A and B.
- Register-file has 3 ports.
    1) Two output-ports allow the contents of 2 different registers to be simultaneously placed on buses A & B.
    2) Third input-port allows data on bus C to be loaded into a third register during the same clock-cycle.
- Buses A and B are used to transfer source-operands to A & B inputs of ALU.
- Result is transferred to destination over bus C.
- Incrementer-unit is used to increment PC by 4.
- Control sequence for the instruction *Add R4,R5,R6* is as follows
    1) PC_{out}, R=B, MAR_{in}, Read, IncPC
    2) WMFC
    3) MDR_{out}, R=B, IR_{in}
    4) $R4_{outA}$, $R5_{outB}$, SelectA, Add, $R6_{in}$, End
- Instruction execution proceeds as follows:

    Step 1--> Contents of PC are passed through ALU using R=B control-signal and loaded into MAR to start a memory Read operation. At the same time, PC is incremented by 4.

    Step2--> Processor waits for MFC signal from memory.

    Step3--> Processor loads requested-data into MDR, and then transfers them to IR.

    Step4--> The instruction is decoded and add operation take place in a single step.

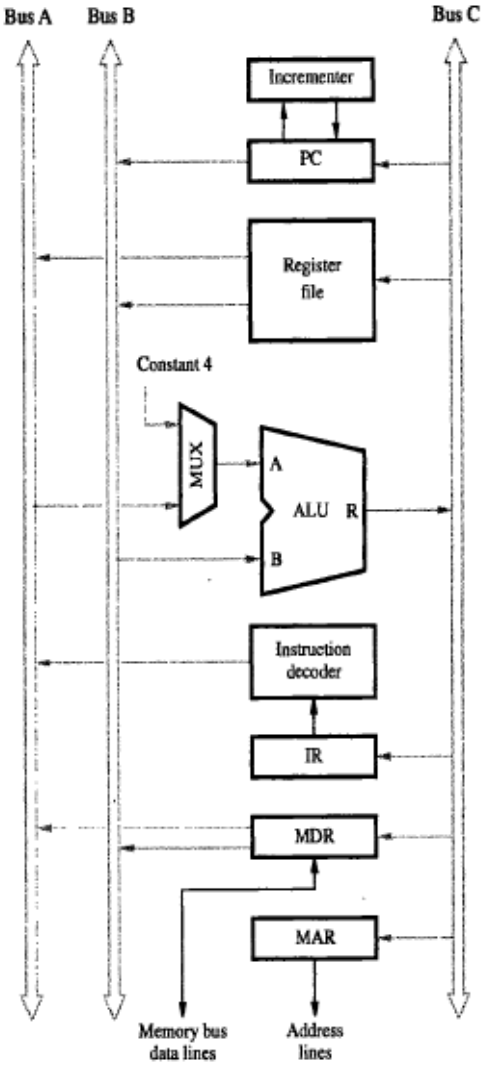


Figure 7.8 Three-bus organization of the datapath.

Note:

To execute instructions, the processor must have some means of generating the control signals needed in the proper sequence. There are two approaches for this purpose:

1) Hardwired control and 2) Microprogrammed control.

HARDWIRED CONTROL

• Decoder/encoder block is a combinational-circuit that generates required control-outputs depending on state of all its inputs.

• Step-decoder provides a separate signal line for each step in the control sequence.
Similarly, output of instruction-decoder consists of a separate line for each machine instruction.

• For any instruction loaded in IR, one of the output-lines INS_1 through INS_m is set to 1, and all other lines are set to 0.

• The input signals to encoder-block are combined to generate the individual control-signals Y_{in}, PC_{out}, Add, End and so on.

• For example, $Z_{in}=T_1+T_6.ADD+T_4.BR$;This signal is asserted during time-slot T_1 for all instructions,
during T_6 for an Add instruction
during T_4 for unconditional branch instruction

• When RUN=1, counter is incremented by 1 at the end of every clock cycle.
When RUN=0, counter stops counting.

• Sequence of operations carried out by this machine is determined by wiring of logic elements, hence the name "hardwired".

• Advantage: Can operate at high speed.
Disadvantage: Limited flexibility.

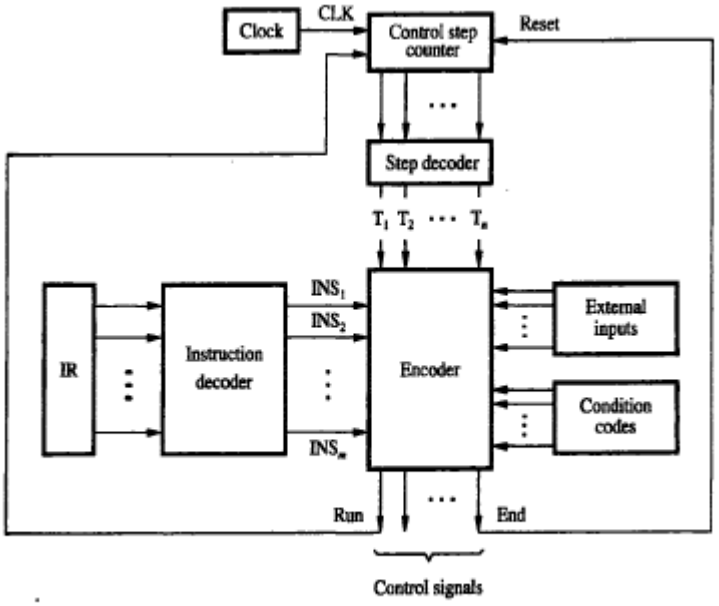


Figure 7.11 Separation of the decoding and encoding functions.

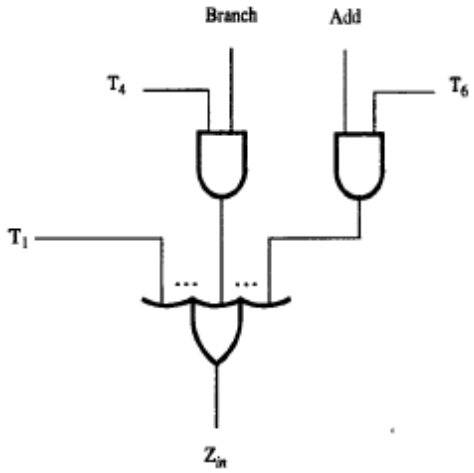


Figure 7.12 Generation of the Z_{in} control signal for the processor in Figure 7.1.

COMPLETE PROCESSOR

- This has separate processing-units to deal with integer data and floating-point data.
- A data-cache is inserted between these processing-units & main-memory.
- Instruction-unit fetches instructions
  - → from an instruction-cache or
  - → from main-memory when desired instructions are not already in cache
- Processor is connected to system-bus & hence to the rest of the computer by means of a bus interface
- Using separate caches for instructions & data is common practice in many processors today.
- A processor may include several units of each type to increase the potential for concurrent operations.

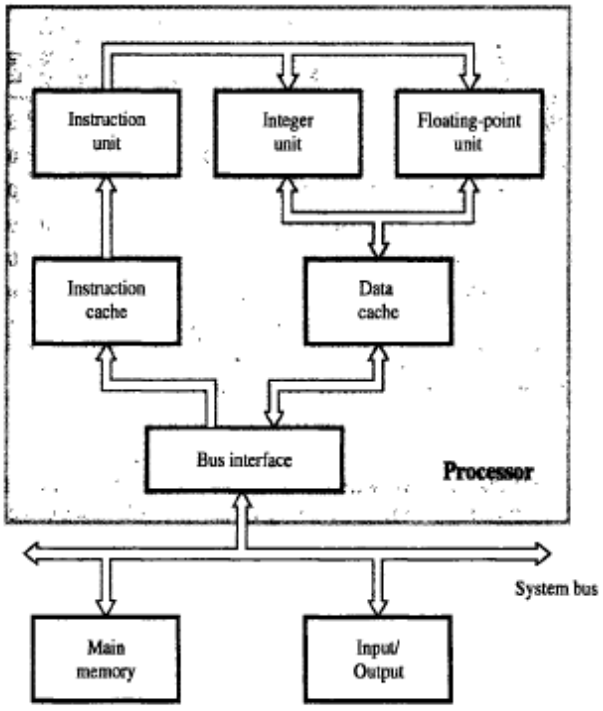


Figure 7.14 Block diagram of a complete processor.

MICROPROGRAMMED CONTROL

• Control-signals are generated by a program similar to machine language programs.

• *Control word(CW)* is a word whose individual bits represent various control-signals(like Add, End, Z_{in}). {Each of the control-steps in control sequence of an instruction defines a unique combination of 1s & 0s in the CW}.

• Individual control-words in microroutine are referred to as *microinstructions.*

• A sequence of CWs corresponding to control-sequence of a machine instruction constitutes the *microroutine*.

• The microroutines for all instructions in the instruction-set of a computer are stored in a special memory called the *control store(CS).*

• Control-unit generates control-signals for any instruction by sequentially reading CWs of corresponding microroutine from CS.

• *Microprogram counter(μPC)* is used to read CWs sequentially from CS.

• Every time a new instruction is loaded into IR, output of "starting address generator" is loaded into μPC.

• Then, μPC is automatically incremented by clock,

causing successive microinstructions to be read from CS.

Hence, control-signals are delivered to various parts of processor in correct sequence.

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.

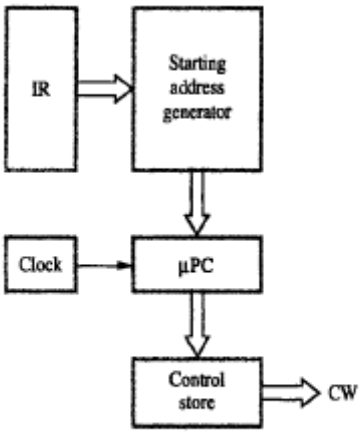


Figure 7.16 Basic organization of a microprogrammed control unit.

ORGANIZATION OF MICROPROGRAMMED CONTROL UNIT (TO SUPPORT CONDITIONAL BRANCHING)

• In case of conditional branching, microinstructions specify which of the external inputs, condition-codes should be checked as a condition for branching to take place.

• The *starting and branch address generator block* loads a new address into μPC when a microinstruction instructs it to do so.

• To allow implementation of a conditional branch, inputs to this block consist of

→ external inputs and condition-codes

→ contents of IR

• μPC is incremented every time a new microinstruction is fetched from microprogram memory except in following situations

i) When a new instruction is loaded into IR, μPC is loaded with starting-address of microroutine for that instruction.

ii) When a Branch microinstruction is encountered and branch condition is satisfied, μPC is loaded with branch-address.

iii) When an End microinstruction is encountered, μPC is loaded with address of first CW in microroutine for instruction fetch cycle.

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.

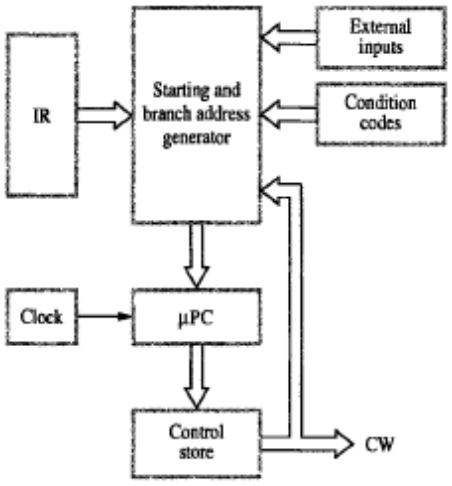


Figure 7.18 Organization of the control unit to allow conditional branching in the microprogram.

MICROINSTRUCTIONS

- Drawbacks of microprogrammed control:
  1) Assigning individual bits to each control-signal results in long microinstructions because the number of required signals is usually large.
  2) Available bit-space is poorly used because only a few bits are set to 1 in any given microinstruction.
- Solution: Signals can be grouped because
  1) Most signals are not needed simultaneously.
  2) Many signals are mutually exclusive.
- Grouping control-signals into fields requires a little more hardware because decoding-circuits must be used to decode bit patterns of each field into individual control signals.
- Advantage: This method results in a smaller control-store (only 20 bits are needed to store the patterns for the 42 signals).

Vertical organization	Horizontal organization
Highly encoded schemes that use compact codes to specify only a small number of control functions in each microinstruction are referred to as a vertical organization	The minimally encoded scheme in which many resources can be controlled with a single microinstuction is called a horizontal organization
This approach results in considerably slower operating speeds because more micrinstructions are needed to perform the desired control functions	This approach is useful when a higher operating speed is desired and when the machine structure allows parallel use of resources

Microinstruction

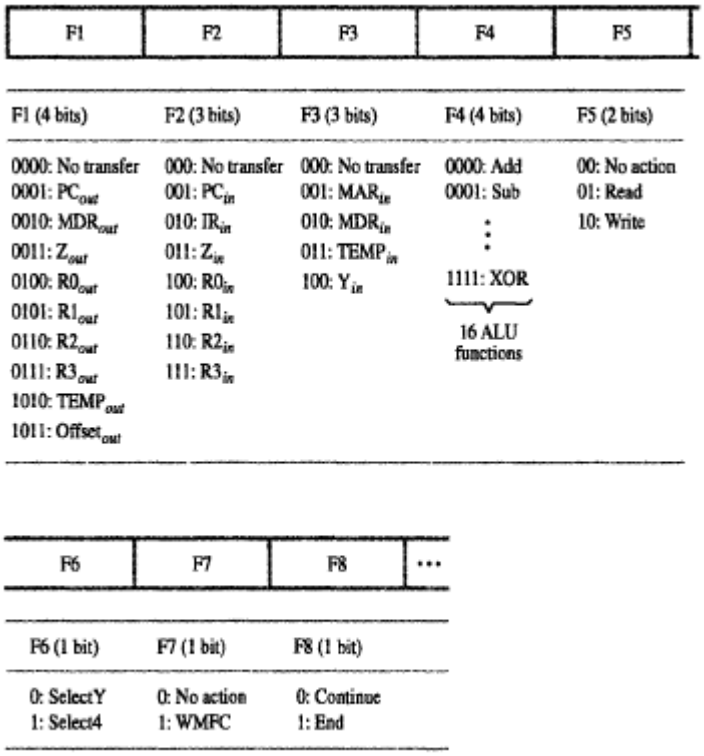


Figure 7.19 An example of a partial format for field-encoded microinstructions.

MICROPROGRAM SEQUENCING

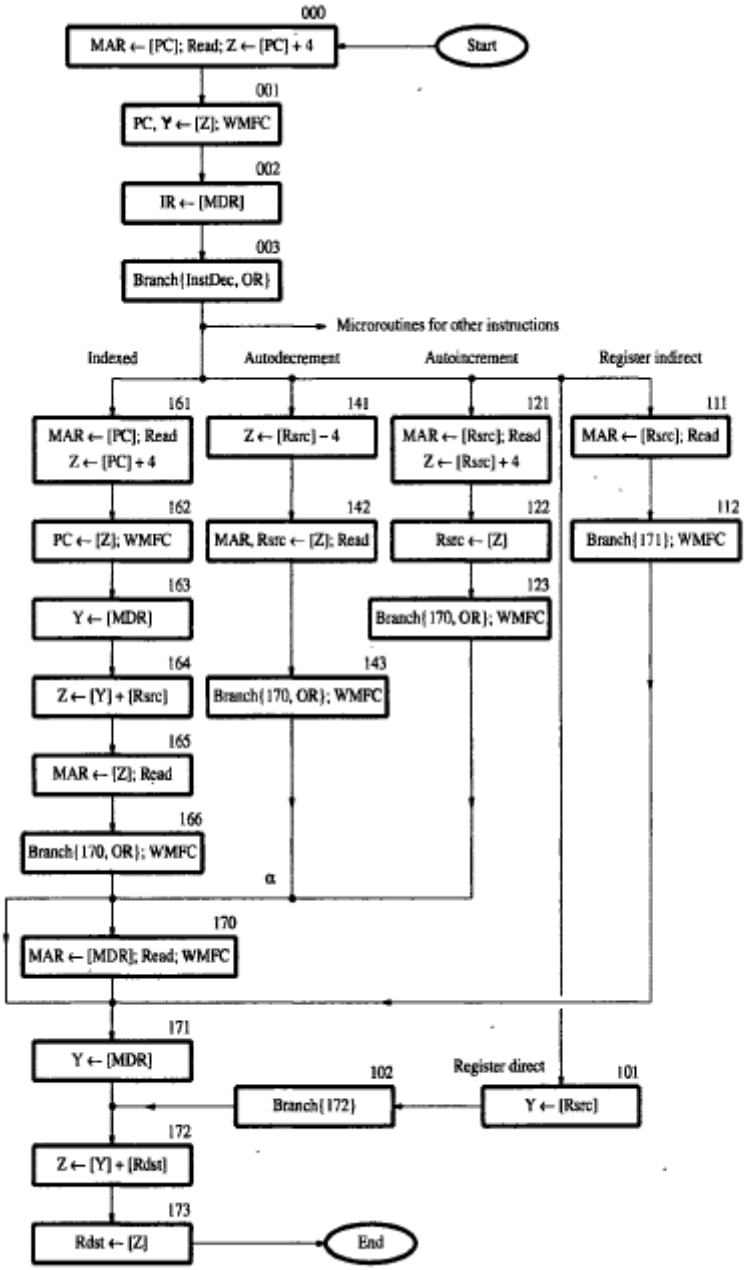


Figure 7.20 Flowchart of a microprogram for the Add src,Rdst instruction.

- Two major disadvantage of microprogrammed control is:
  1) Having a separate microroutine for each machine instruction results in a large total number of microinstructions and a large control-store.
  2) Execution time is longer because it takes more time to carry out the required branches.
- Consider the instruction *Add src,Rdst* ;which adds the source-operand to the contents of Rdst and places the sum in Rdst.
- Let source-operand can be specified in following addressing modes: register, autoincrement, autodecrement and indexed as well as the indirect forms of these 4 modes.
- Each box in the chart corresponds to a microinstruction that controls the transfers and operations indicated within the box.
- The microinstruction is located at the address indicated by the octal number (001,002).

BRANCH ADDRESS MODIFICATION USING BIT-ORING

• Consider the point labeled α in the figure. At this point, it is necessary to choose between direct and indirect addressing modes.

• If indirect-mode is specified in the instruction, then the microinstruction in location 170 is performed to fetch the operand from the memory.

If direct-mode is specified, this fetch must be bypassed by branching immediately to location 171.

• The most efficient way to bypass microinstruction 170 is to have the preceding branch microinstructions specify the address 170 and then use an OR gate to change the LSB of this address to 1 if the direct addressing mode is involved. This is known as the *bit-ORing* technique.

WIDE BRANCH ADDRESSING

• The instruction-decoder(InstDec) generates the starting-address of the microroutine that implements the instruction that has just been loaded into the IR.

• Here, register IR contains the Add instruction, for which the instruction decoder generates the microinstruction address 101. (However, this address cannot be loaded as is into the μPC).

• The source-operand can be specified in any of several addressing-modes. The bit-ORing technique can be used to modify the starting-address generated by the instruction-decoder to reach the appropriate path.

Use of WMFC

• WMFC signal is issued at location 112 which causes a branch to the microinstruction in location 171.

• WMFC signal means that the microinstruction may take several clock cycles to complete. If the branch is allowed to happen in the first clock cycle, the microinstruction at location 171 would be fetched and executed prematurely. To avoid this problem, WMFC signal must inhibit any change in the contents of the μPC during the waiting-period.

Detailed Examination

• Consider *Add (Rsrc)+,Rdst;* which adds Rsrc content to Rdst content, then stores the sum in Rdst and finally increments Rsrc by 4 (i.e. auto-increment mode).

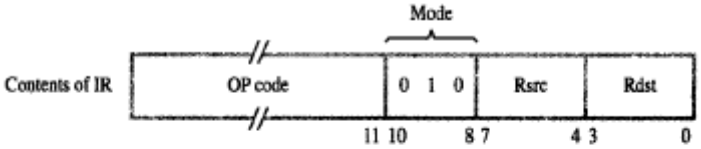
• In bit 10 and 9, bit-patterns 11, 10, 01 and 00 denote indexed, auto-decrement, auto-increment and register modes respectively. For each of these modes, bit 8 is used to specify the indirect version.

• The processor has 16 registers that can be used for addressing purposes; each specified using a 4-bit-code.

• There are 2 stages of decoding:

1) The microinstruction field must be decoded to determine that an Rsrc or Rdst register is involved.

2) The decoded output is then used to gate the contents of the Rsrc or Rdst fields in the IR into a second decoder, which produces the gating-signals for the actual registers R0 to R15.



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 ← 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 ← 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 FIELDS

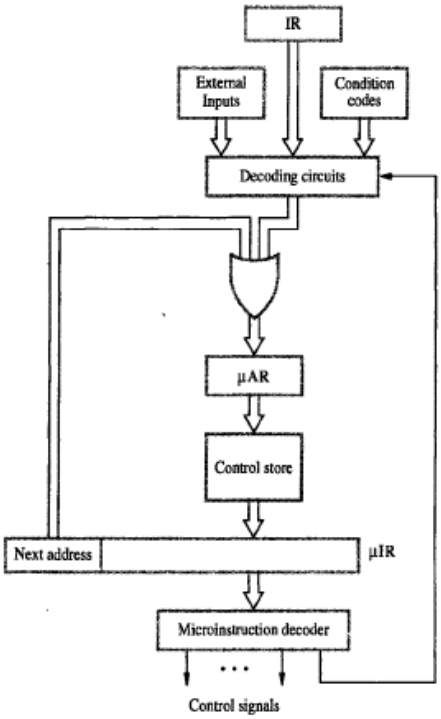


Figure 7.22 Microinstruction-sequencing organization.

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 7.5.3.

Octal address	F0	F1	F2	F3	F4	F5	F6	F7	F8	F9	F10
000	00000001	001	011	001	0000	01	1	0	0	0	0
001	00000010	011	001	100	0000	00	0	1	0	0	0
002	00000011	010	010	000	0000	00	0	0	0	0	0
003	00000000	000	000	000	0000	00	0	0	1	1	0
121	01010010	100	011	001	0000	01	1	0	0	0	0
122	01111000	011	100	000	0000	00	0	1	0	0	1
170	01111001	010	000	001	0000	01	0	1	0	0	0
171	01111010	010	000	100	0000	00	0	0	0	0	0
172	01111011	101	011	000	0000	00	0	0	0	0	0
173	00000000	011	101	000	0000	00	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.)

• The microprogram requires several branch microinstructions which perform no useful operation. Thus, they detract from the operating speed of the computer.
• Solution: Include an address-field as a part of every microinstruction to indicate the location of the next microinstruction to be fetched. (This means every microinstruction becomes a branch microinstruction).
• The flexibility of this approach comes at the expense of additional bits for the address-field.
• Advantage: Separate branch microinstructions are virtually eliminated. There are few limitations in assigning addresses to microinstructions. There is no need for a counter to keep track of sequential addresse. Hence, the μPC is replaced with a μAR (Microinstruction Address Register). {which is loaded from the next-address field in each microinstruction}.
• The next-address bits are fed through the OR gate to the μAR, so that the address can be modified on the basis of the data in the IR, external inputs and condition-codes.
• The decoding circuits generate the starting-address of a given microroutine on the basis of the opcode in the IR.

PREFETCHING MICROINSTRUCTIONS

• Drawback of microprogrammed control: Slower operating speed because of the time it takes to fetch microinstructions from the control-store.

• Solution: Faster operation is achieved if the next microinstruction is pre-fetched while the current one is being executed.

Emulation

• The main function of microprogrammed control is to provide a means for simple, flexible and relatively inexpensive execution of machine instruction.

• Its flexibility in using a machine's resources allows diverse classes of instructions to be implemented.

• Suppose we add to the instruction-repository of a given computer M1, an entirely new set of instructions that is in fact the instruction-set of a different computer M2.

• Programs written in the machine language of M2 can be then be run on computer M1 i.e. M1 emulates M2.

• Emulation allows us to replace obsolete equipment with more up-to-date machines.

• If the replacement computer fully emulates the original one, then no software changes have to be made to run existing programs.

• Emulation is easiest when the machines involved have similar architectures.