Control Unit

Hamacher:

Basic Components of Processor

- ALU
- Control Unit
- Registers
- Internal bus
- External bus
  - The registers, ALU, and interconnecting bus are collectively referred to as **datapath**

Functions of Components

- ALU
  - performs arithmetic and logic operations on data values stored in registers or memory
- CU
  - Controls the sequence of all operations

  i.e. **instruction sequencing**
  - Activates the specific hardware units that are required for the set of operations that execution of a machine instruction requires

  i.e. **instruction decoding**

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).

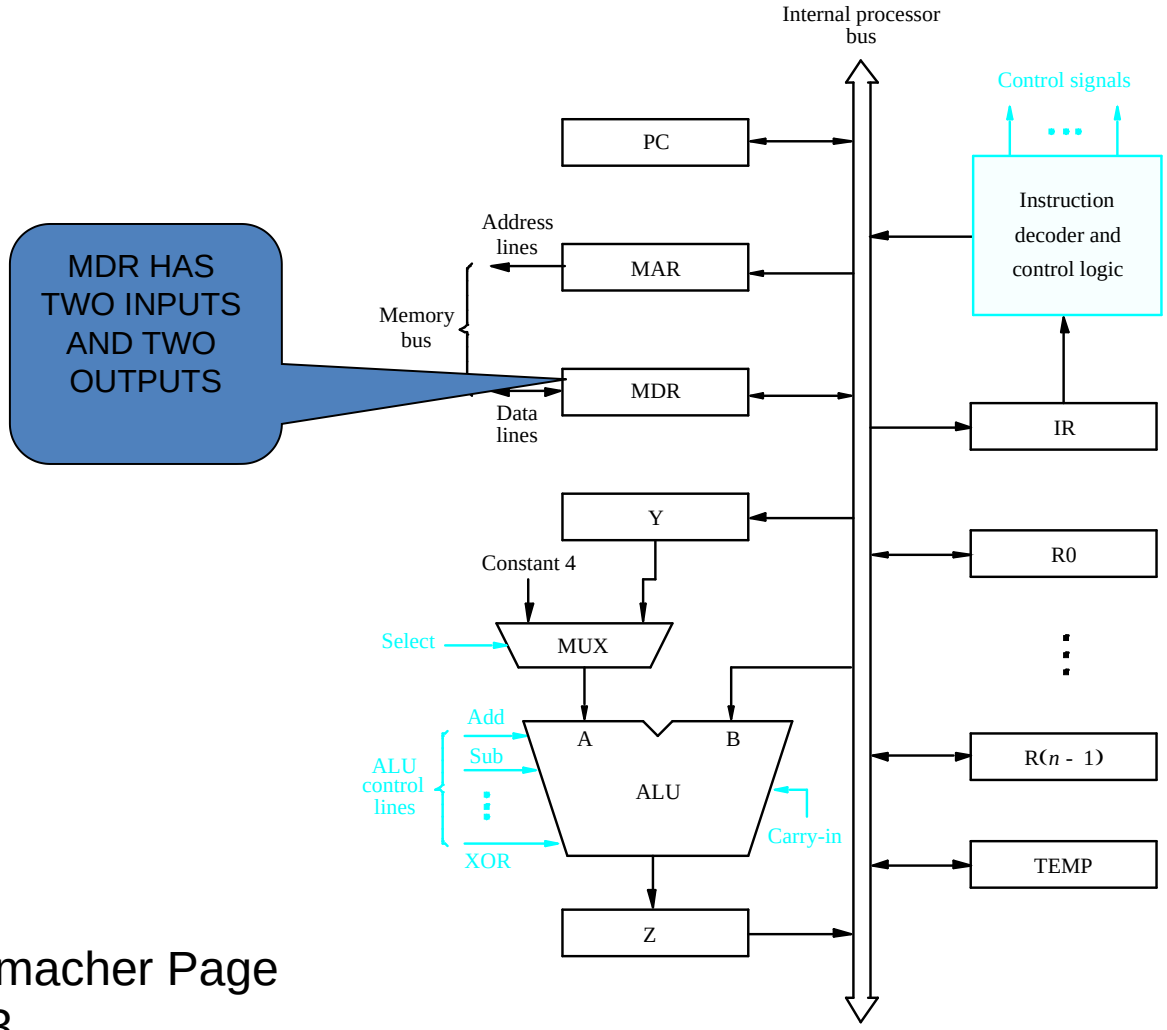
  $$IR \leftarrow [[PC]]$$

- 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

Hamacher Page 413

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

Executing a Microoperation

- 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

Register Transfers

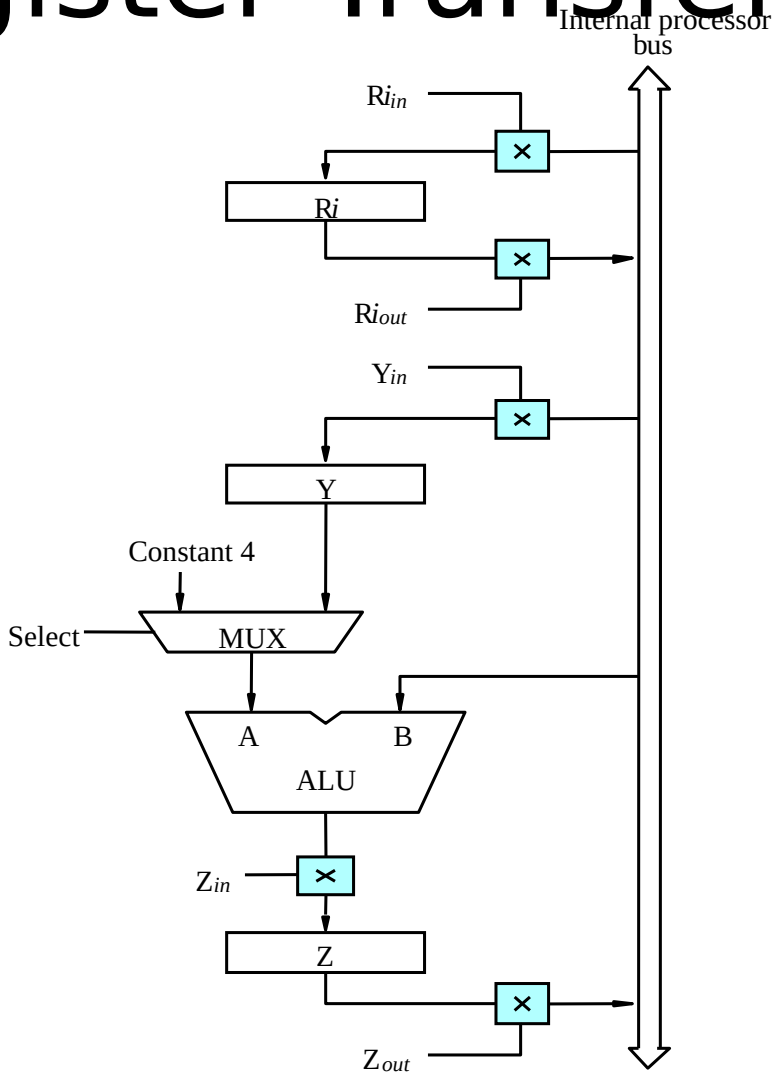


Figure 7.2. Input and output gating for the registers in Figure 7.1.

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

Register Transfers

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

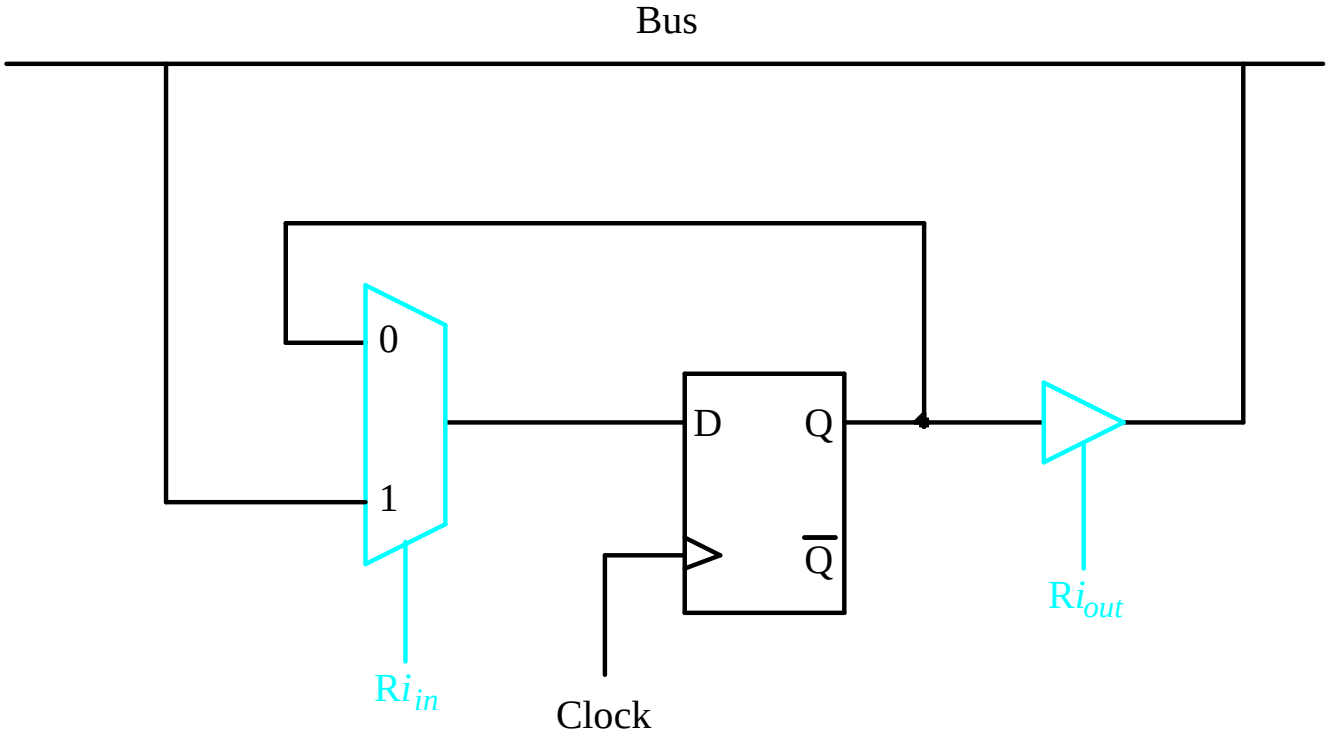


Figure 7.3. Input and output gating for one register bit.

Fetching a Word from Memory

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

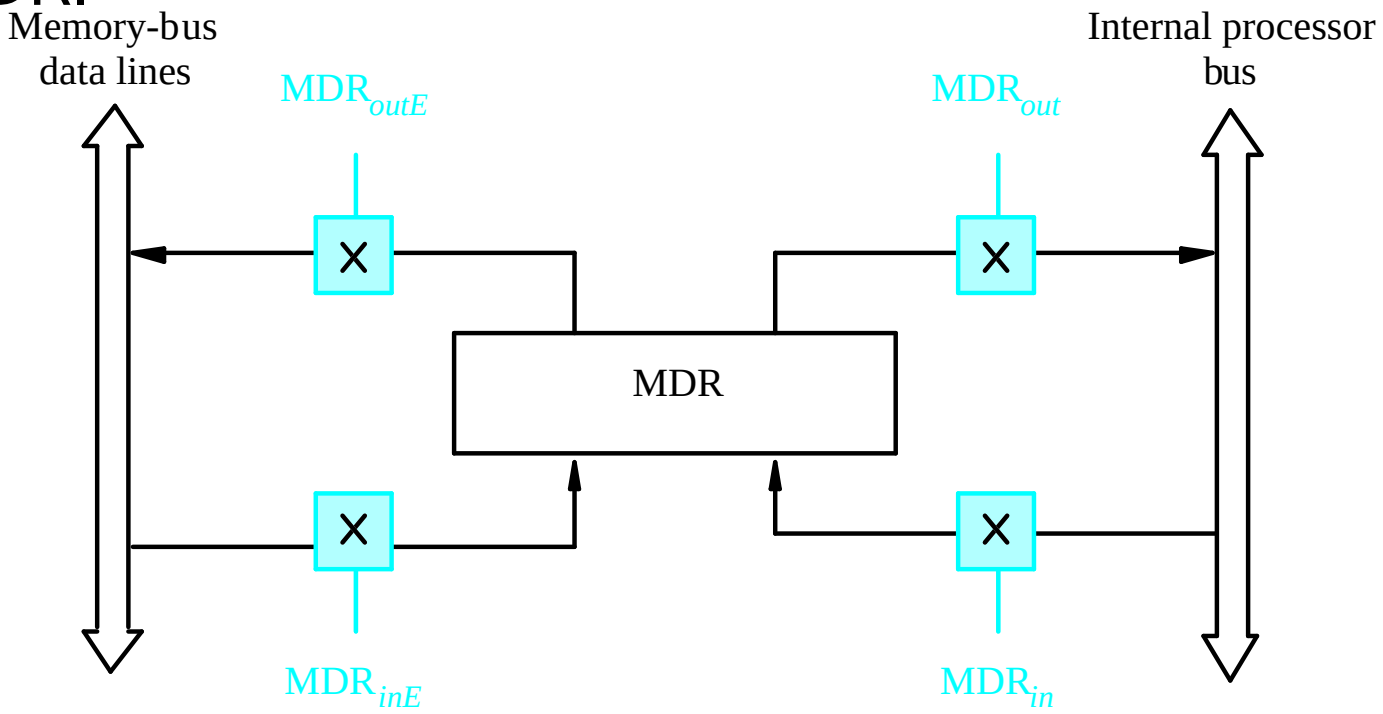


Figure 7.4. Connection and control signals for register MDR.

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]

Execution of a Complete Instruction

- Add (R3), R1
- Fetch the instruction
- (PC=PC + 4 (Memory byte addressable,4 byte word)
- 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

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.

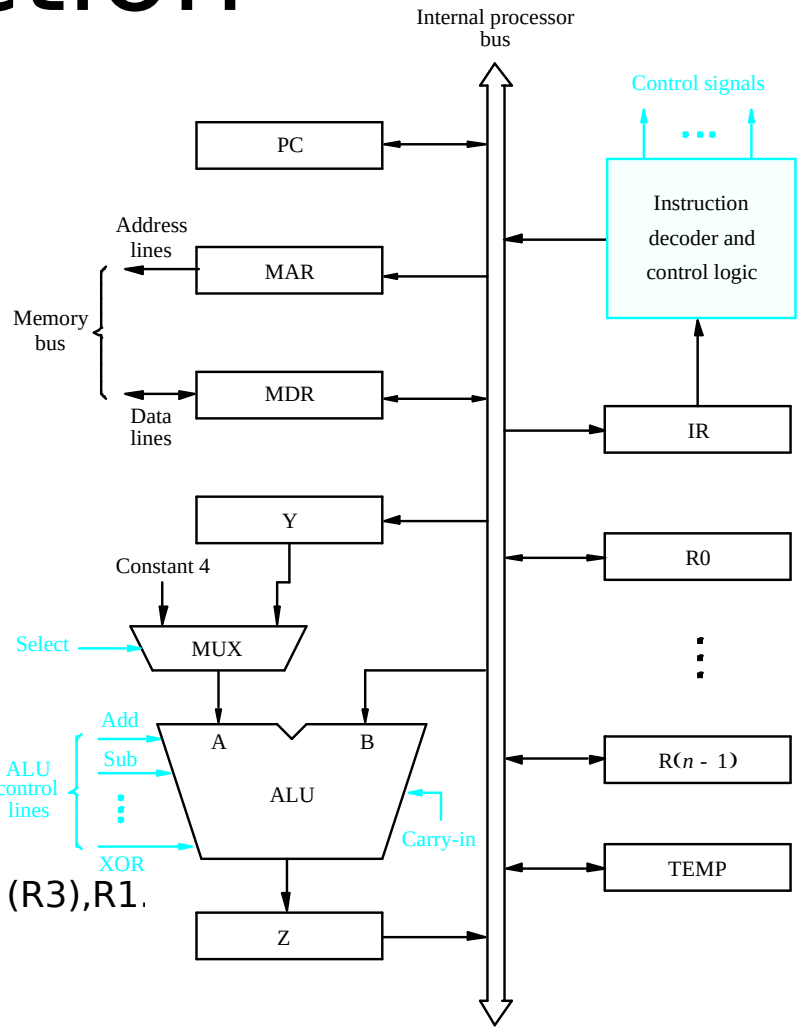


Figure 7.1. Single-bus organization of the datapath inside a processor.

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	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.

- What is the control sequence for execution of the instruction

  Add R1, R2

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

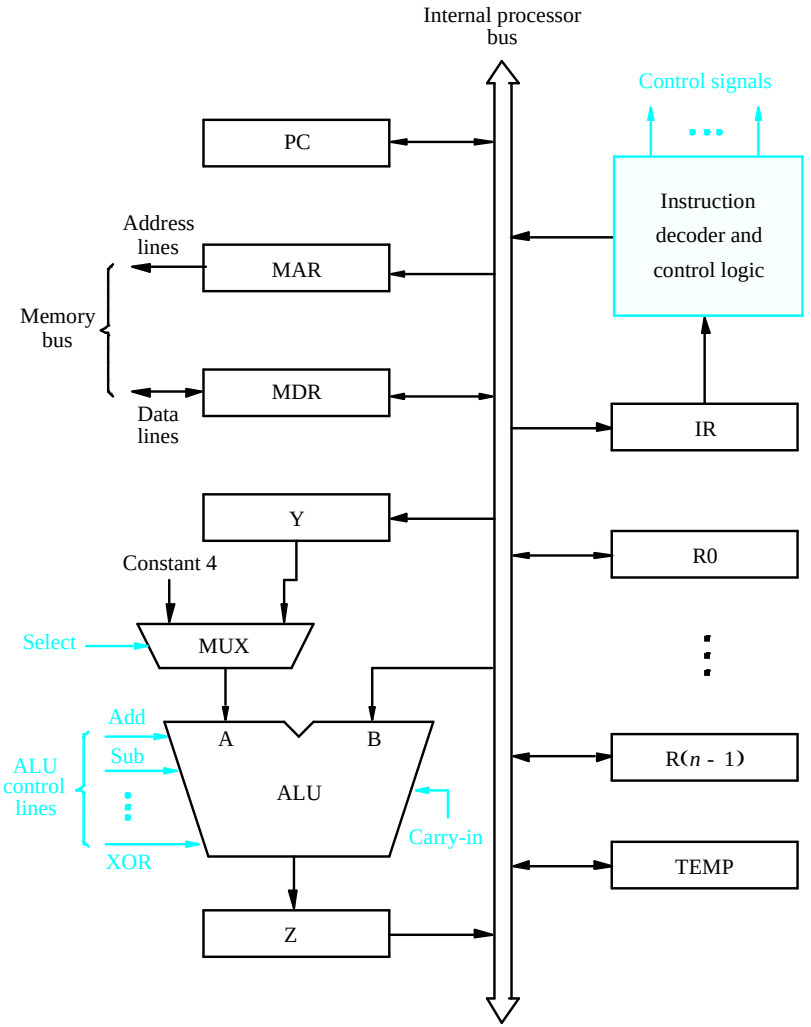


Figure 7.1. Single-bus organization of the datapath inside a processor.

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
  - microprogrammed control

Hardwired Control

- **Hardwired system can operate at high speed; but with little flexibility.**
- **Involves the use of fixed instruction**
- **Fixed logic blocks, encoders and decoders etc**
- **High speed operation**
- **Expensive**
- **Relatively Complex**

Control Unit Organization

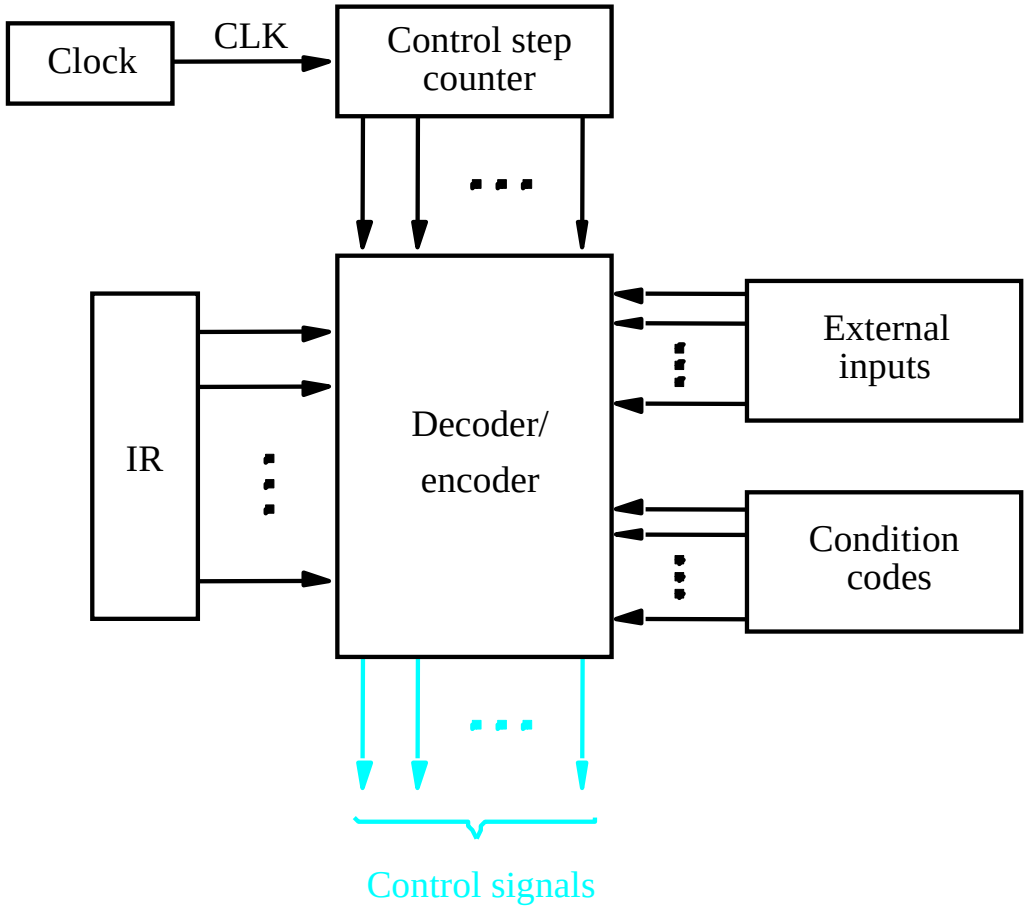


Figure 7.10. Control unit organization.

Detailed Block Description

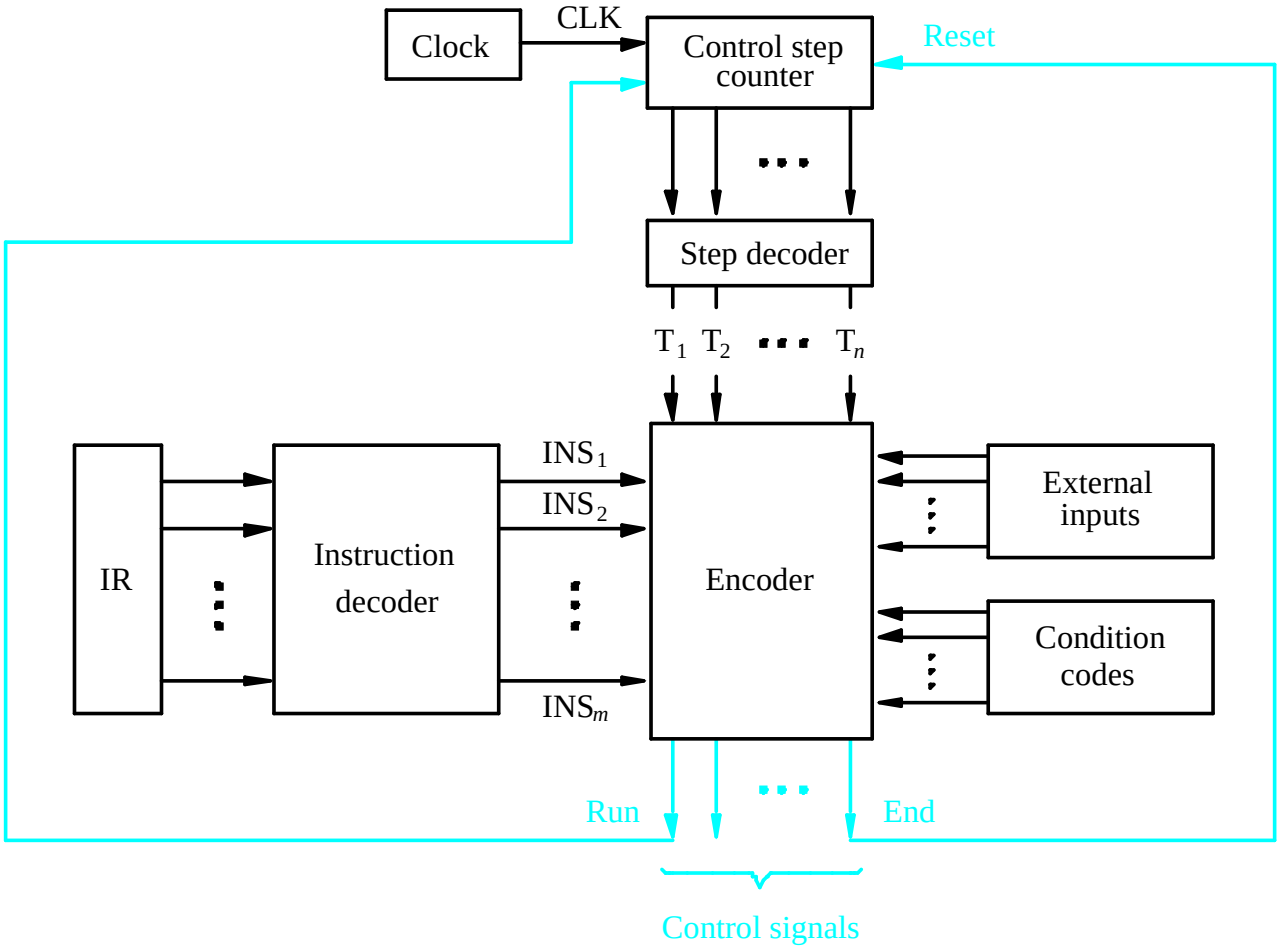


Figure 7.11. Separation of the decoding and encoding functions.

Microprogrammed Control Unit

TERMINOLOGY

Microprogram

- **Program stored in memory that generates all the control signals required to execute the instruction set correctly**
- **Consists of microinstructions**

Microinstruction

- **Contains a control word and a sequencing word**
  Control Word - All the control information required for one clock cycle
  Sequencing Word - Information needed to decide the next microinstruction address
- **Vocabulary to write a microprogram**

Control Memory(Control Storage: CS)

- **Storage in the microprogrammed control unit to store the microprogram**

Writeable Control Memory(Writeable Control Storage:WCS)

- **CS whose contents can be modified**
  -> Allows the microprogram can be changed
  -> Instruction set can be changed or modified

Dynamic Microprogramming

- **Computer system whose control unit is implemented with a microprogram in WCS**
- **Microprogram can be changed by a systems programmer or a user**

Hardwired Control Unit Vs Microprogrammed Control Unit

Hardwired Control Unit	Microprogrammed Control Unit
Control logic is implemented with flags, decoders, logic gates and other digital circuits.	**Control logic is the instructions that are stored in control memory to initiate the required sequence of micro-operation.**
It is a hardware control unit.	**It is a mid-way between hardware and software.**
It is faster and known to have complex structure	It is comparatively slow and is simple in structure.
It is difficult to design, test and implement.	**It is easy to design, test and implement.**
It is inflexible to modify.	**It is flexible to modify.**
It is expensive and high error.	**It is cheaper and less error.**
It is used in RISC processor.	**It is used in CISC processor**

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}	$R1_{out}$	$R1_{in}$	$R3_{out}$	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.

Overview

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.

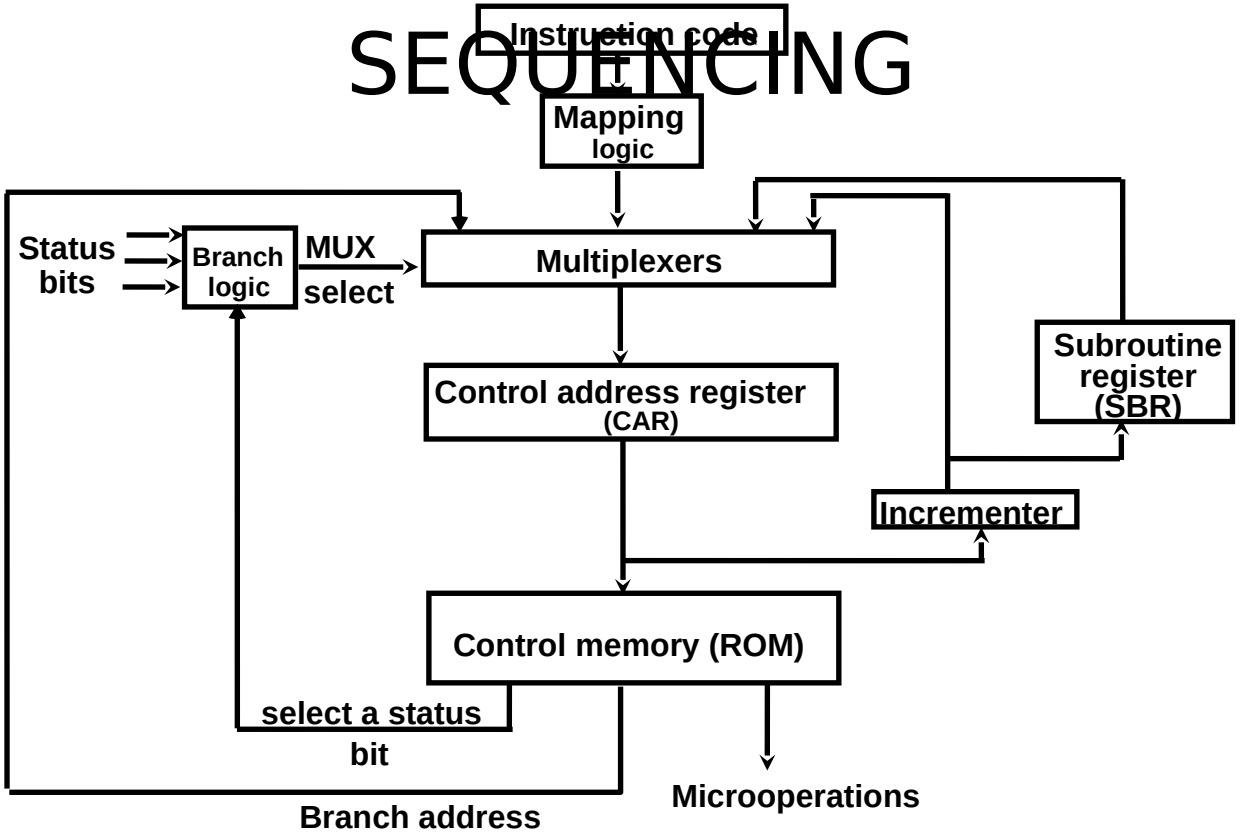
TERMINOLOGY

Sequencer (Microprogram Sequencer)

A Microprogram Control Unit that determines the Microinstruction Address to be executed in the next clock cycle

- **In-line Sequencing**
- **Branch**
- **Conditional Branch**
- **Subroutine**
- **Loop**
- **Instruction OP-code mapping**

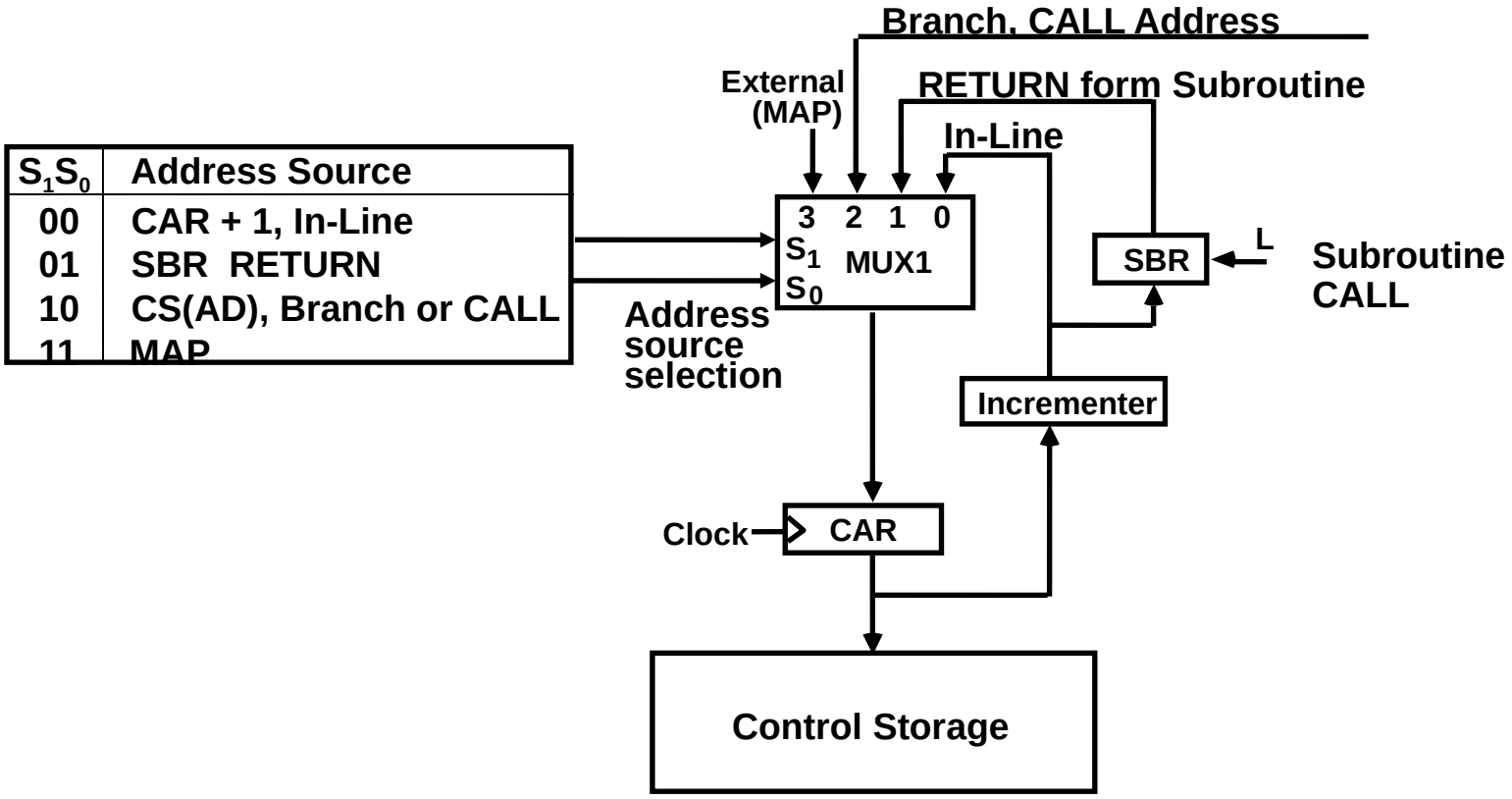
MICROINSTRUCTION SEQUENCING



Sequencing Capabilities Required in a Control Storage

- Incrementing of the control address register
- Unconditional and conditional branches
- A mapping process from the bits of the machine instruction to an address for control memory
- A facility for subroutine call and return

MICROPROGRAM SEQUENCER
- NEXT MICROINSTRUCTION ADDRESS LOGIC -

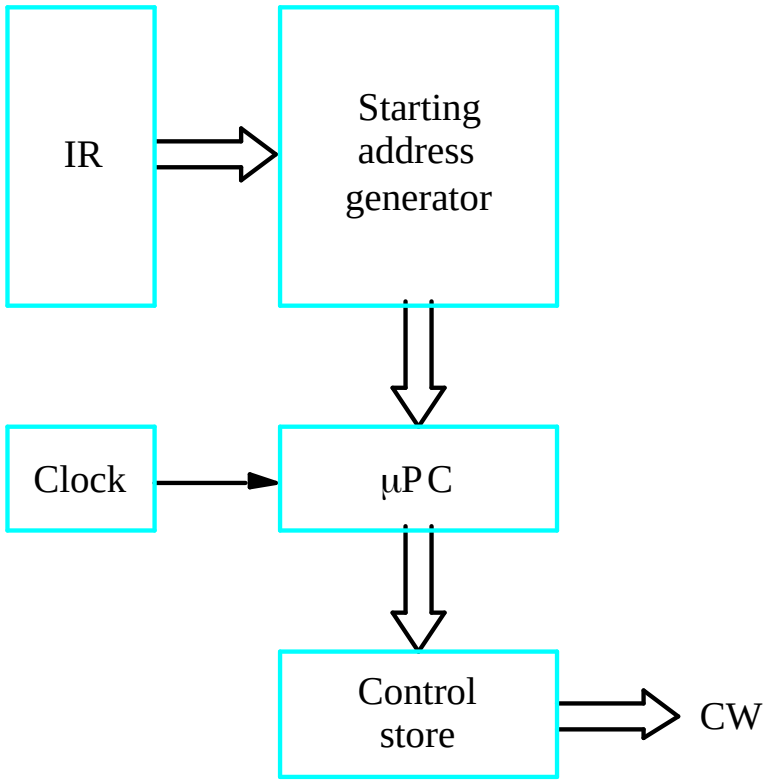


MUX-1 selects an address from one of four sources and routes it into a CAR

- **In-Line Sequencing → CAR + 1**
- **Branch, Subroutine Call → CS(AD)**
- **Return from Subroutine → Output of SBR**
- **New Machine instruction → MAP**

Overview

- Control store



One function cannot be carried out by this simple organization.

Figure 7.16. Basic organization of a microprogrammed control unit.

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.

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.

Overview

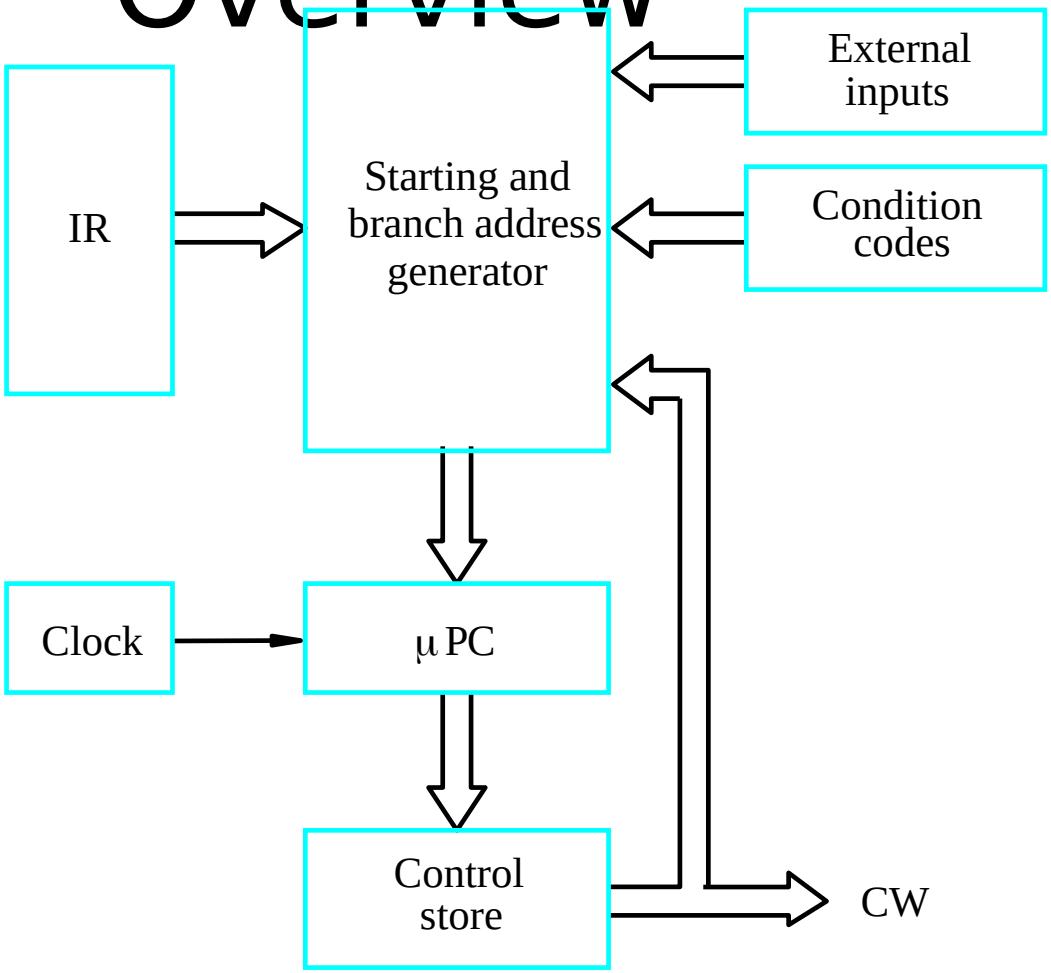
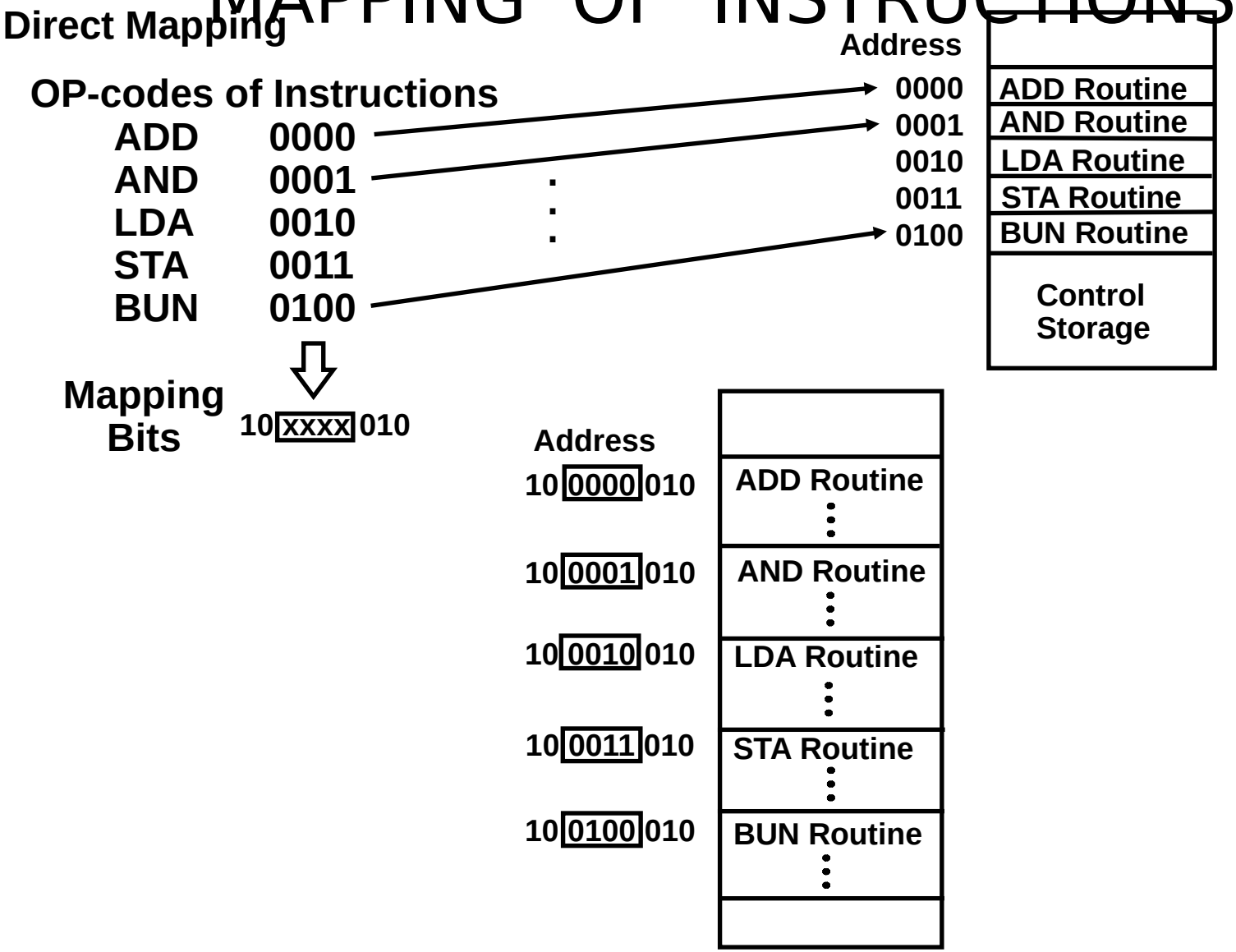


Figure 7.18. Organization of the control unit to allow conditional branching in the microprogram.

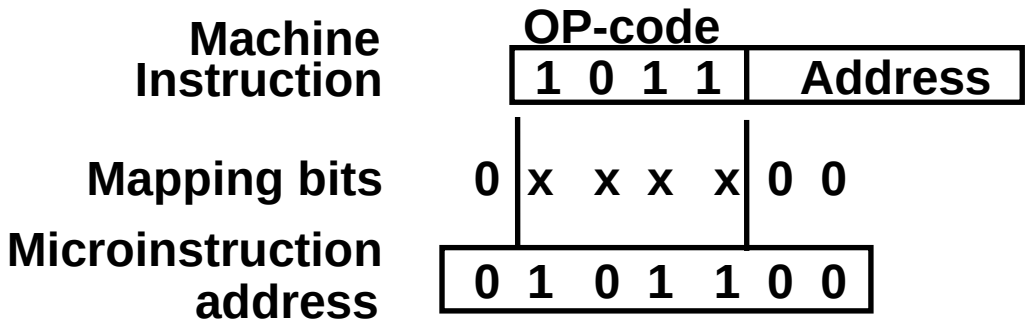
MAPPING OF INSTRUCTIONS

Direct Mapping

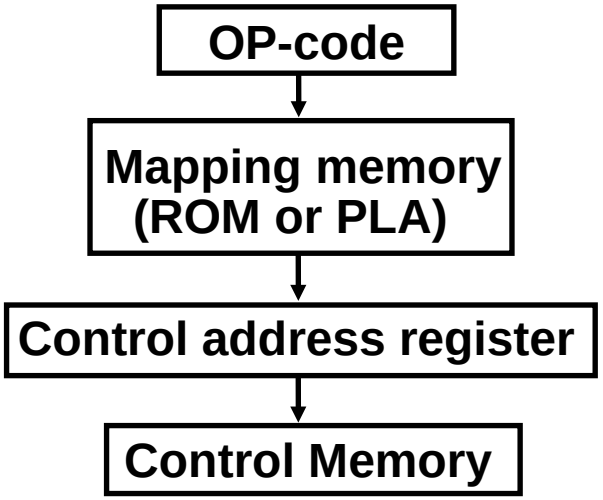


MAPPING OF INSTRUCTIONS TO MICROROUTINES

Mapping from the OP-code of an instruction to the address of the Microinstruction which is the starting microinstruction of its execution microprogram

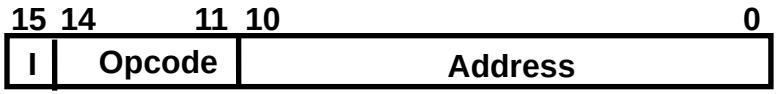


Mapping function implemented by ROM or PLA



MACHINE INSTRUCTION FORMAT

Machine instruction format

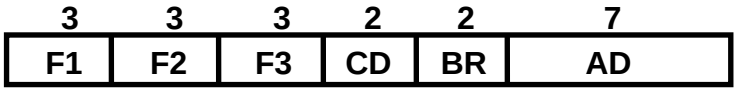


Sample machine instructions

Symbol	OP-code	Description
ADD	0000	AC ← AC + M[EA]
BRANCH	0001	if (AC < 0) then (PC ← EA)
STORE	0010	M[EA] ← AC
EXCHANGE	0011	AC ← M[EA], M[EA] ← AC

EA is the effective address

Microinstruction Format



F1, F2, F3: Microoperation fields
CD: Condition for branching
BR: Branch field
AD: Address field

MICROINSTRUCTION FIELD DESCRIPTIONS - F1,F2,F3

F1	Microoperation	Symbol
000	None	NOP
001	AC ← AC + DR	ADD
010	AC ← 0	CLRAC
011	AC ← AC + 1	INCAC
100	AC ← DR	DRTAC
101	AR ← DR(0-10)	DRTAR
110	AR ← PC	PCTAR
111	M[AR] ← DR	WRITE

F2	Microoperation	Symbol
000	None	NOP
001	AC ← AC - DR	SUB
010	AC ← AC ∨ DR	OR
011	AC ← AC ∧ DR	AND
100	DR ← M[AR]	READ
101	DR ← AC	ACTDR
110	DR ← DR + 1	INCDR
111	DR(0-10) ← PC	PCTDR

F3	Microoperation	Symbol
000	None	NOP
001	AC ← AC ⊕ DR	XOR
010	AC ← AC’	COM
011	AC ← shl AC	SHL
100	AC ← shr AC	SHR
101	PC ← PC + 1	INCPC
110	PC ← AR	ARTPC
111	Reserved	

MICROINSTRUCTION FIELD DESCRIPTIONS - CD, BR

CD	Condition	Symbol	Comments
00	Always = 1	U	Unconditional branch
01	DR(15)	I	Indirect address bit
10	AC(15)	S	Sign bit of AC
11	AC = 0	Z	Zero value in AC

BR	Symbol	Function
00	JMP	CAR ← AD if condition = 1
		CAR ← CAR + 1 if condition = 0
01	CALL	CAR ← AD, SBR ← CAR + 1 if condition = 1
		CAR ← CAR + 1 if condition = 0
10	RET	CAR ← SBR (Return from subroutine)
11	MAP	CAR(2-5) ← DR(11-14), CAR(0,1,6) ← 0

SYMBOLIC MICROINSTRUCTIONS

- **Symbols are used in microinstructions as in assembly language**
- **A symbolic microprogram can be translated into its binary equivalent by a microprogram assembler.**

Sample Format

five fields: label; micro-ops; CD; BR; AD

Label: may be empty or may specify a symbolic address terminated with a colon

Micro-ops: consists of one, two, or three symbols separated by commas

CD: one of {U, I, S, Z}, where U: Unconditional Branch
I: Indirect address bit
S: Sign of AC
Z: Zero value in AC

BR: one of {JMP, CALL, RET, MAP}

AD: one of {Symbolic address, NEXT, empty}

SYMBOLIC MICROPROGRAM - FETCH ROUTINE

During FETCH, Read an instruction from memory and decode the instruction and update PC

Sequence of microoperations in the fetch cycle:

```
AR ← PC
DR ← M[AR], PC ← PC + 1
AR ← DR(0-10), CAR(2-5) ← DR(11-14), CAR(0,1,6) ← 0
```

Symbolic microprogram for the fetch cycle:

```
        ORG 64
FETCH:  PCTAR          U   JMP   NEXT
        READ, INCPC    U   JMP   NEXT
        DRTAR          U   MAP
```

Binary equivalents translated by an assembler

Binary address	F1	F2	F3	CD	BR	AD
1000000	110	000	000	00	00	1000001
1000001	000	100	101	00	00	1000010
1000010	101	000	000	00	11	0000000