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).

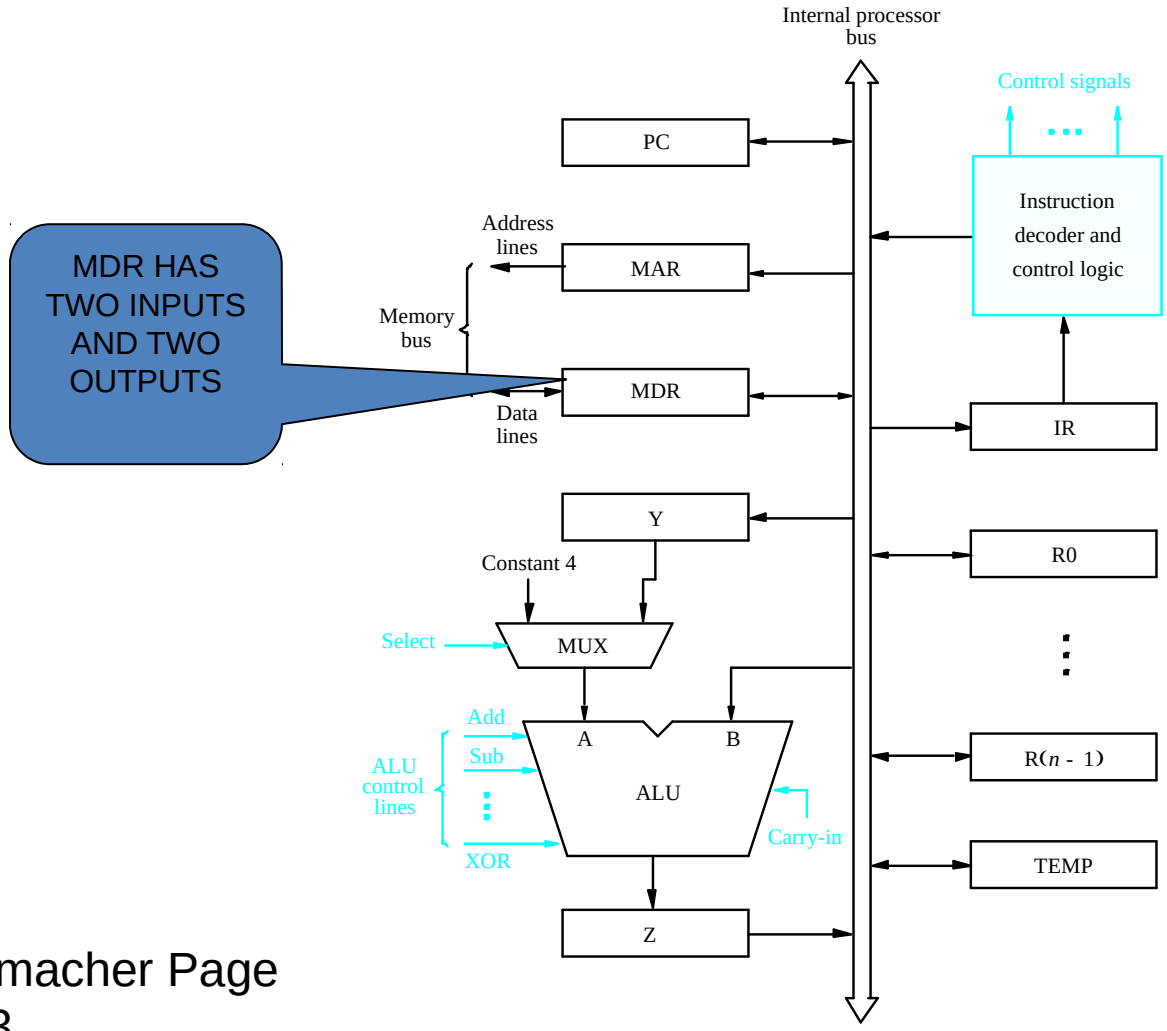
  $$IR \leftarrow [[PC]]$$

- 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

- 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

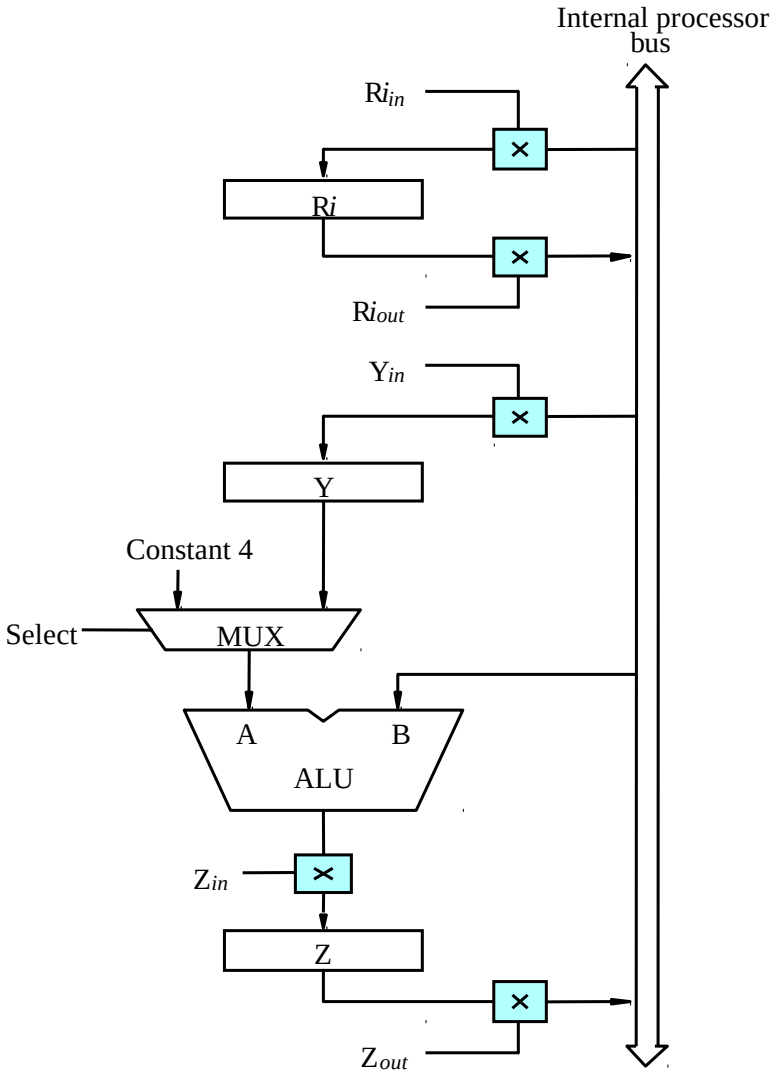


Figure 7.2. Input and output gating for the registers in Figure 7.1.

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.

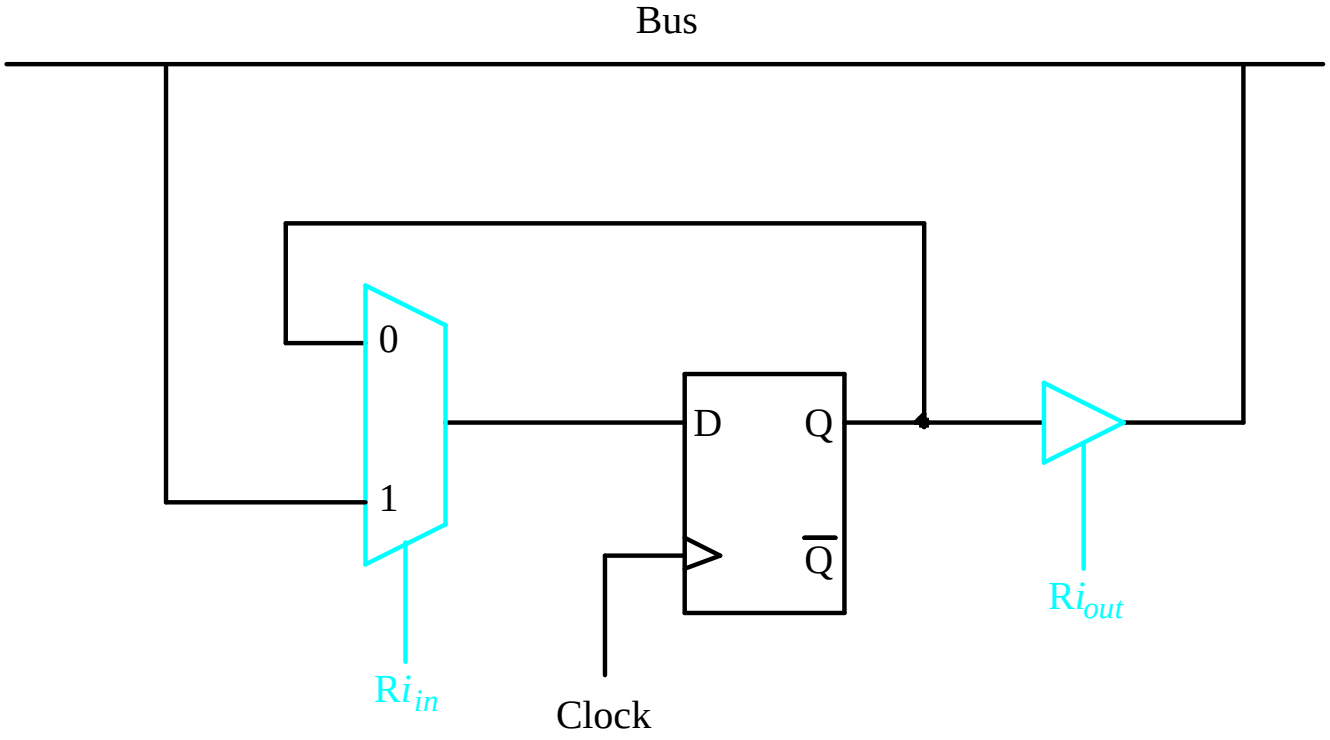


Figure 7.3. Input and output gating for one register bit.

Fetching a Word from Memory

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

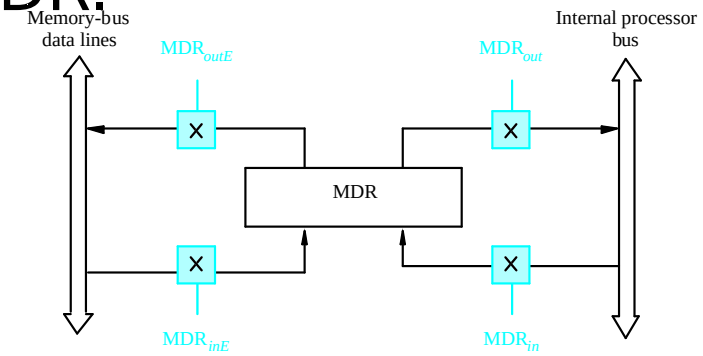


Figure 7.4. Connection and control signals for register MDR.

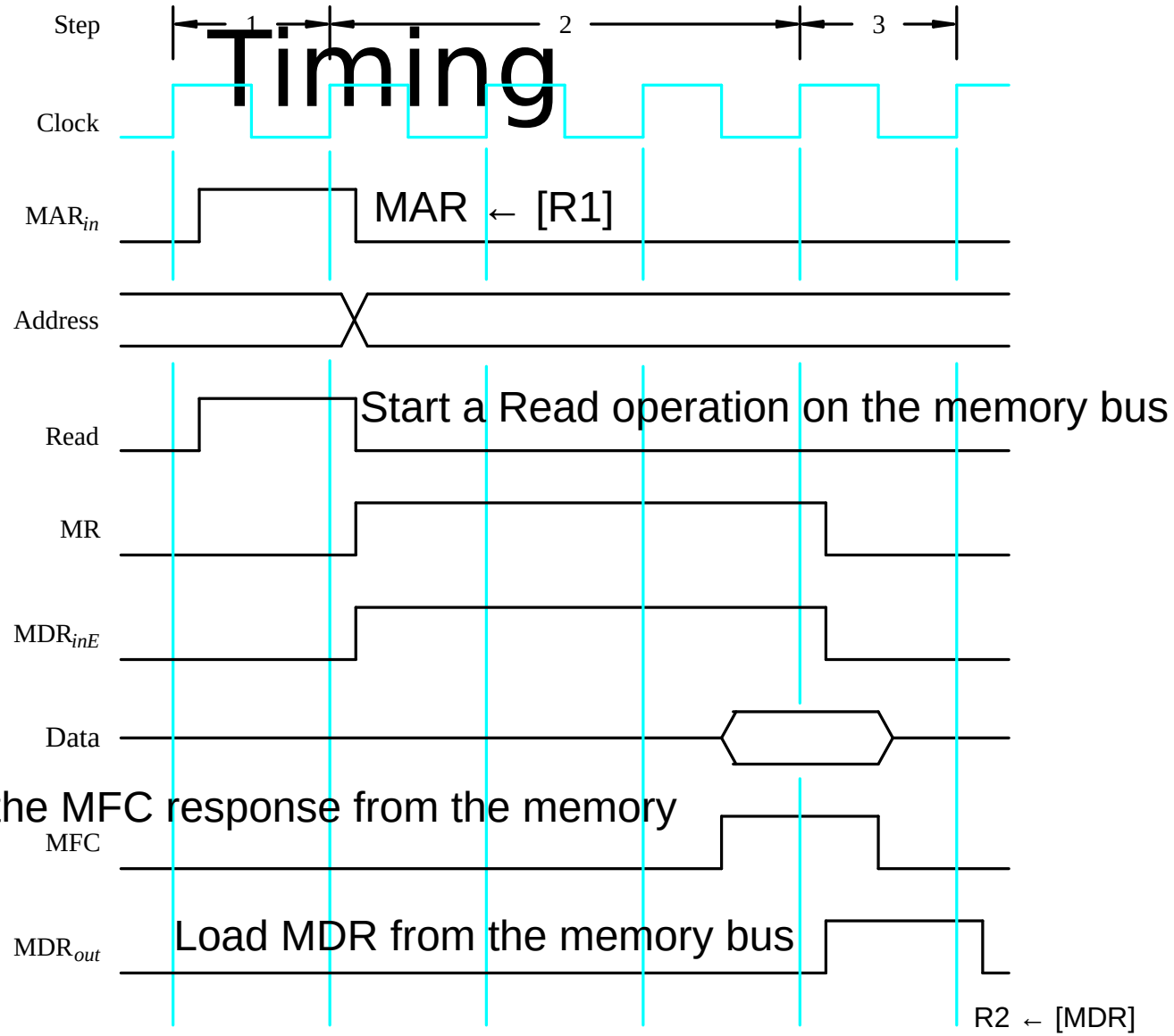
Figure 7.4. Connection and control signals for register MDR.

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]

Timing

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



Timing of a memory Read operation.

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

Architecture

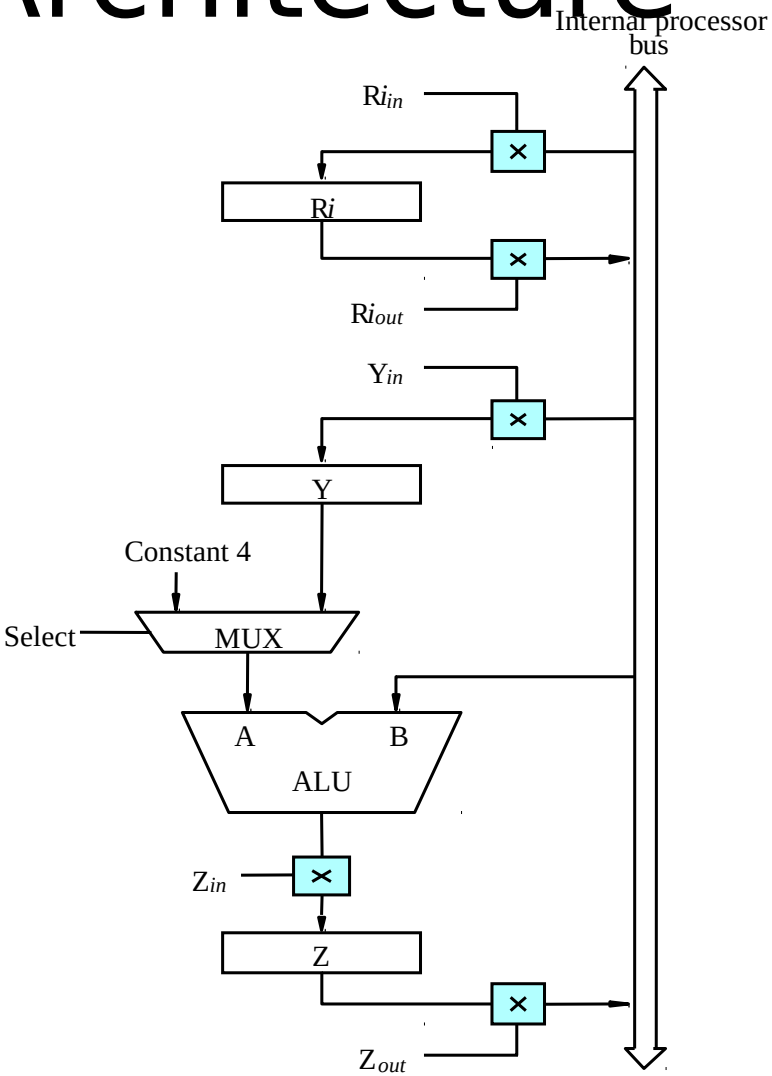


Figure 7.2. Input and output gating for the registers in Figure 7.1.

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.

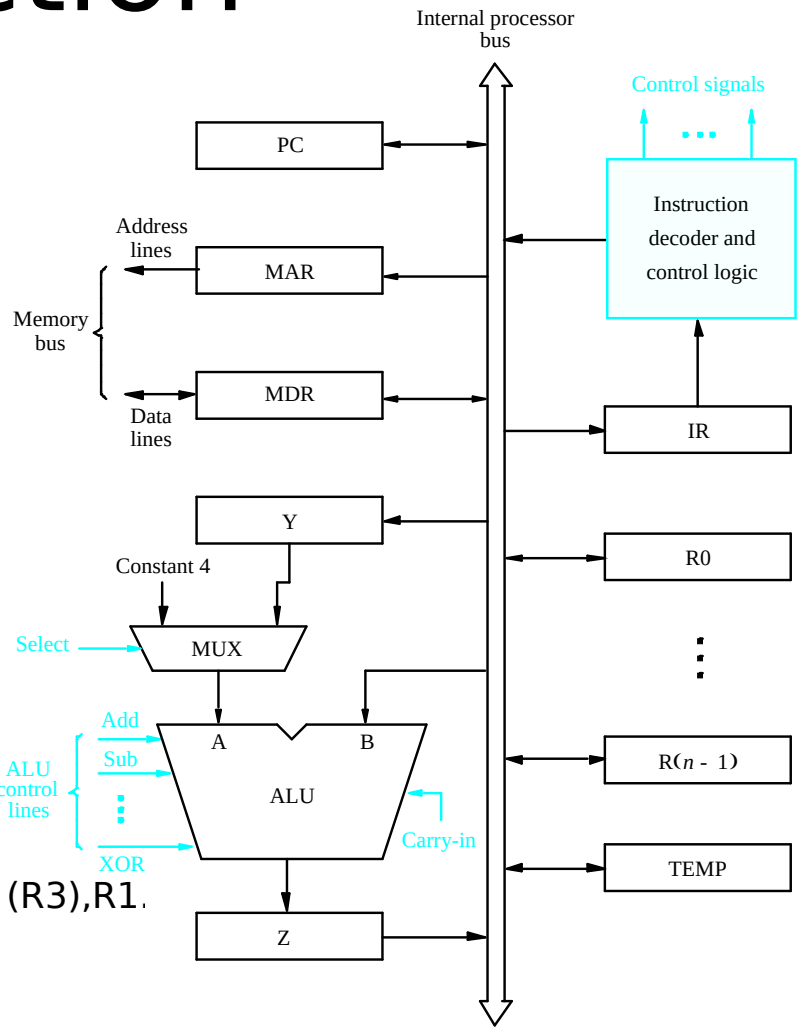


Figure 7.1. Single-bus organization of the datapath inside a processor.

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)

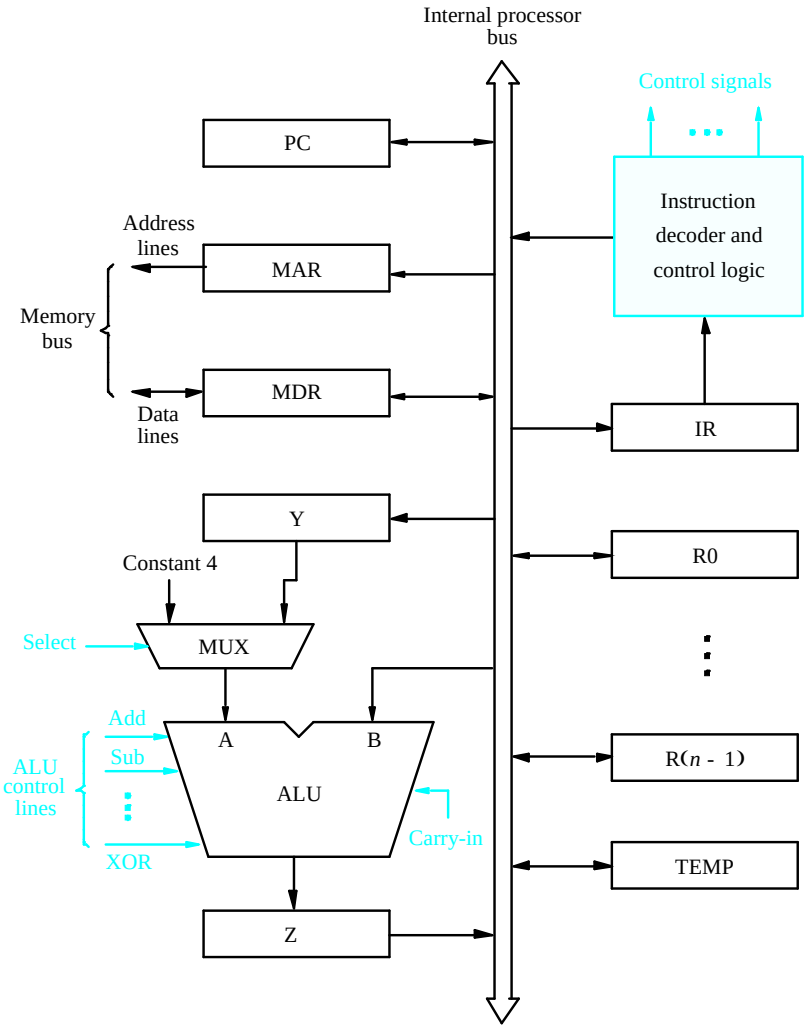


Figure 7.1. Single-bus organization of the datapath inside a processor.

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

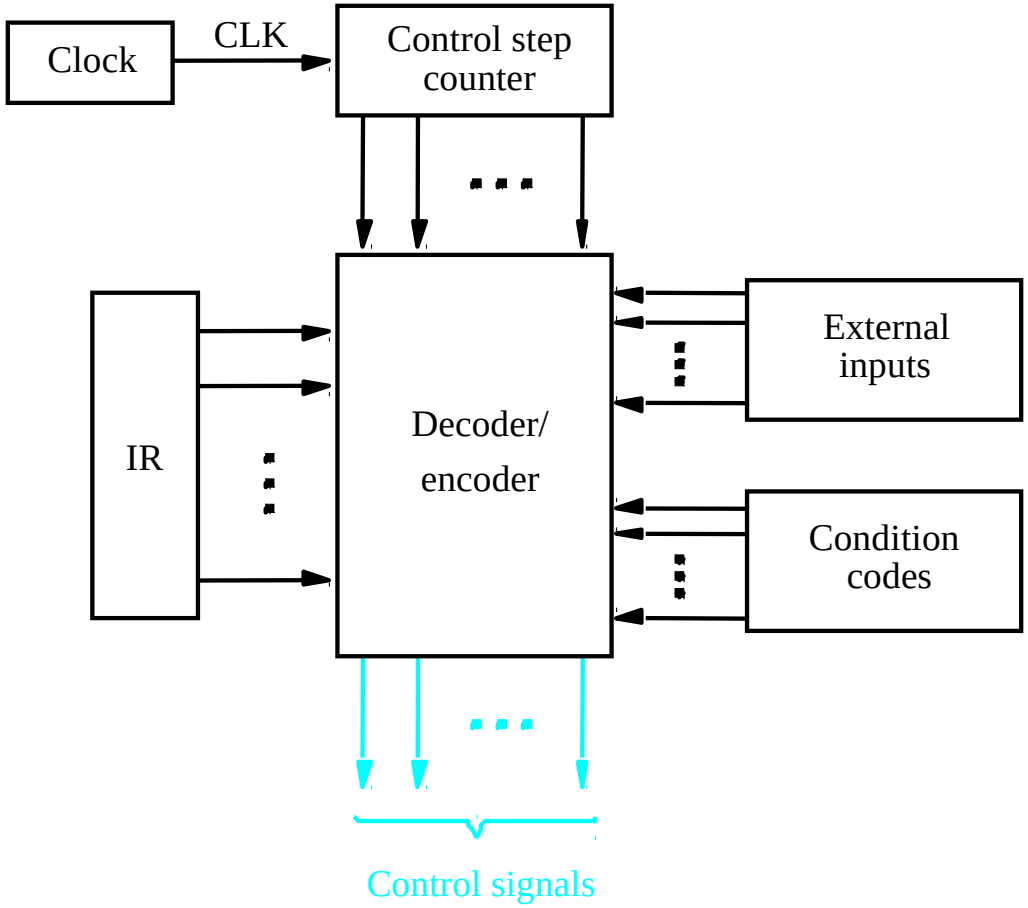


Figure 7.10. Control unit organization.

Detailed Block Description

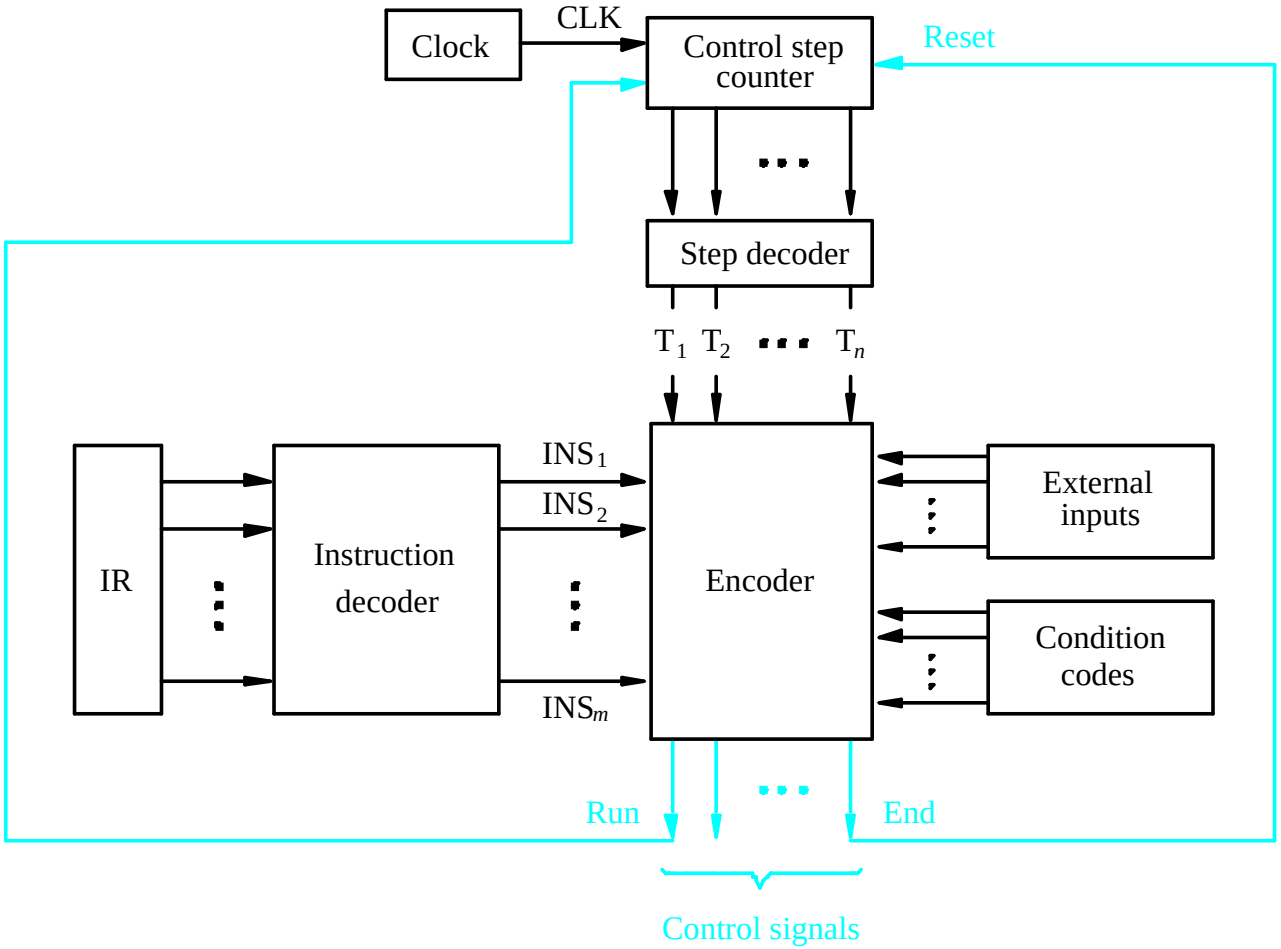


Figure 7.11. Separation of the decoding and encoding functions.

Generating Z_{in}

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

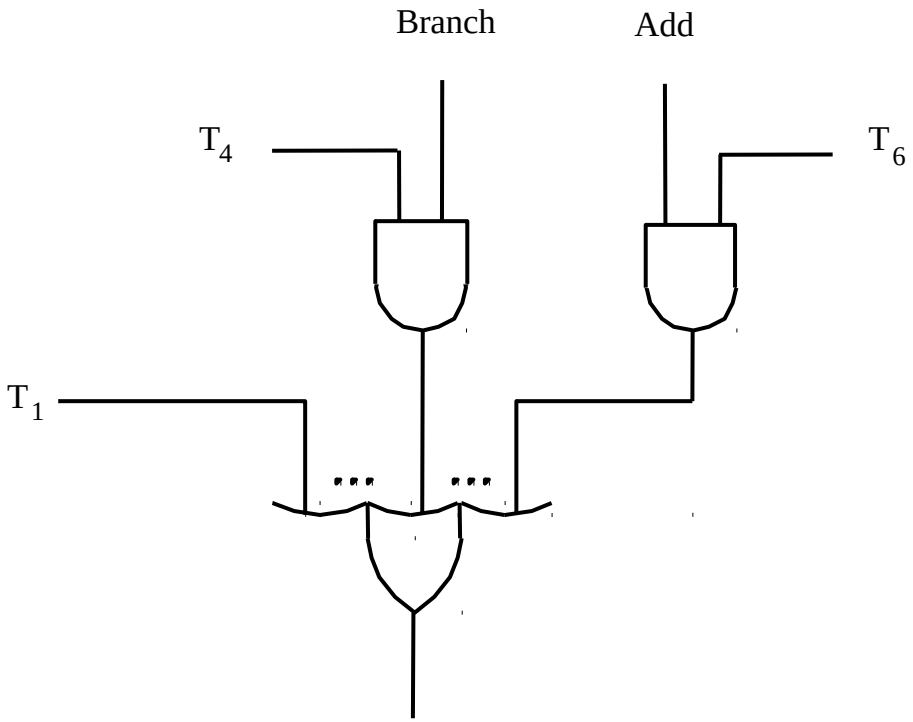


Figure 7.12. Generation of the Z_{in} control signal for the processor in Figure 7.1.

Generating End

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

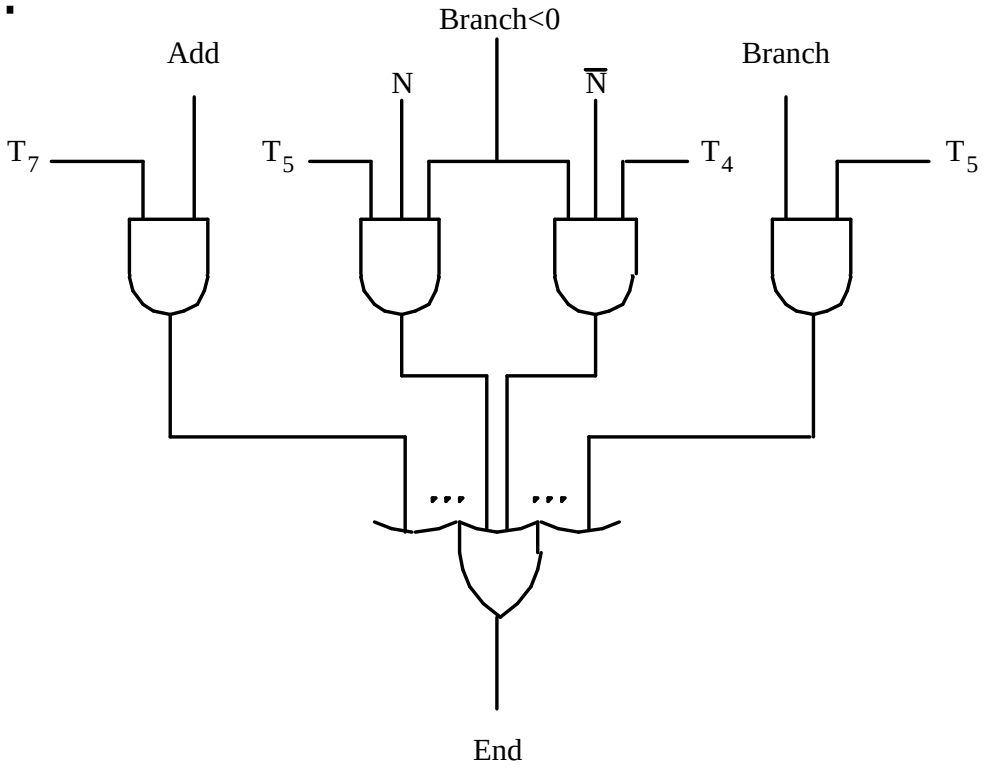


Figure 7.13. Generation of the End control signal.

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**

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.

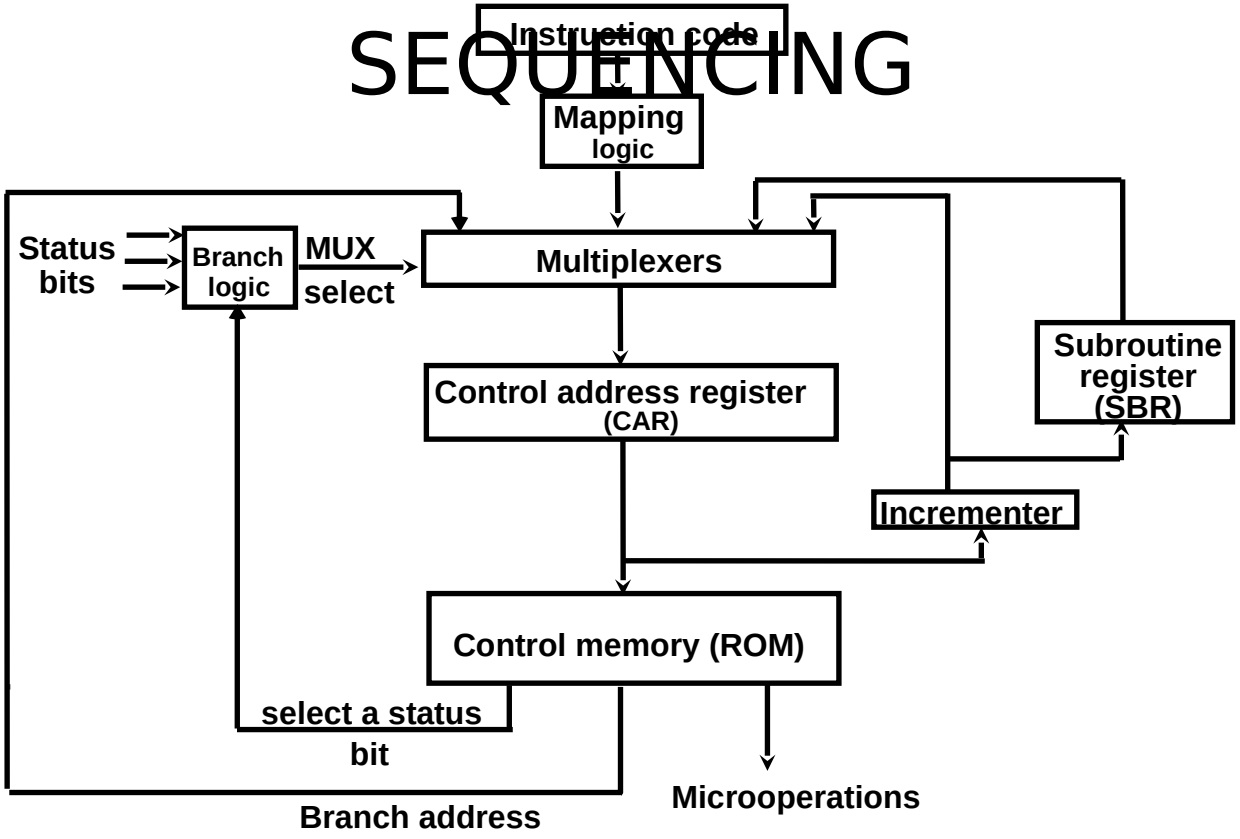
TERMINOLOGY

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

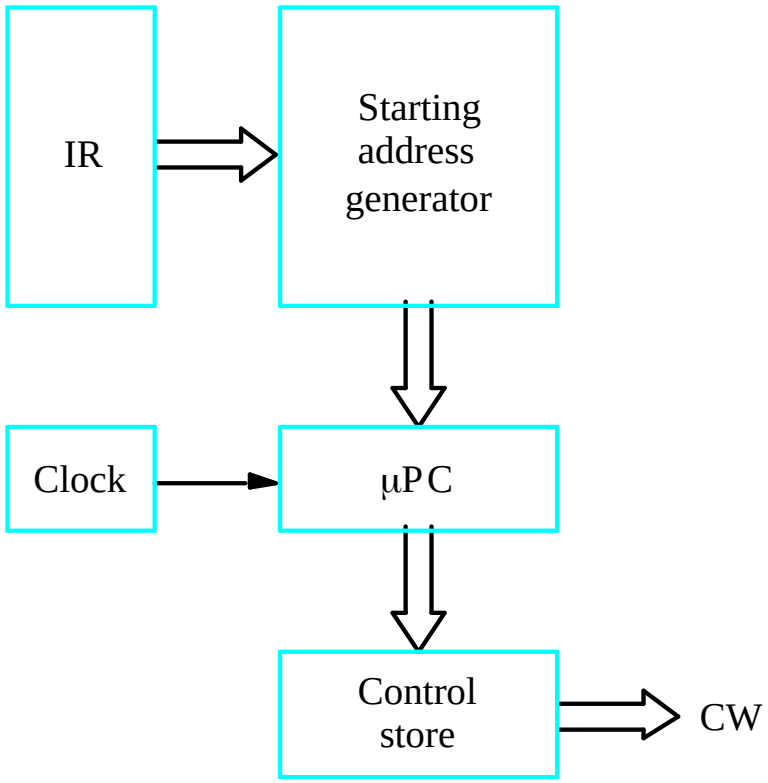


Sequencing Capabilities Required in a Control Storage

- 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

Overview

- Control store



One function cannot be carried out by this simple organization.

Figure 7.16. Basic organization of a microprogrammed control unit.

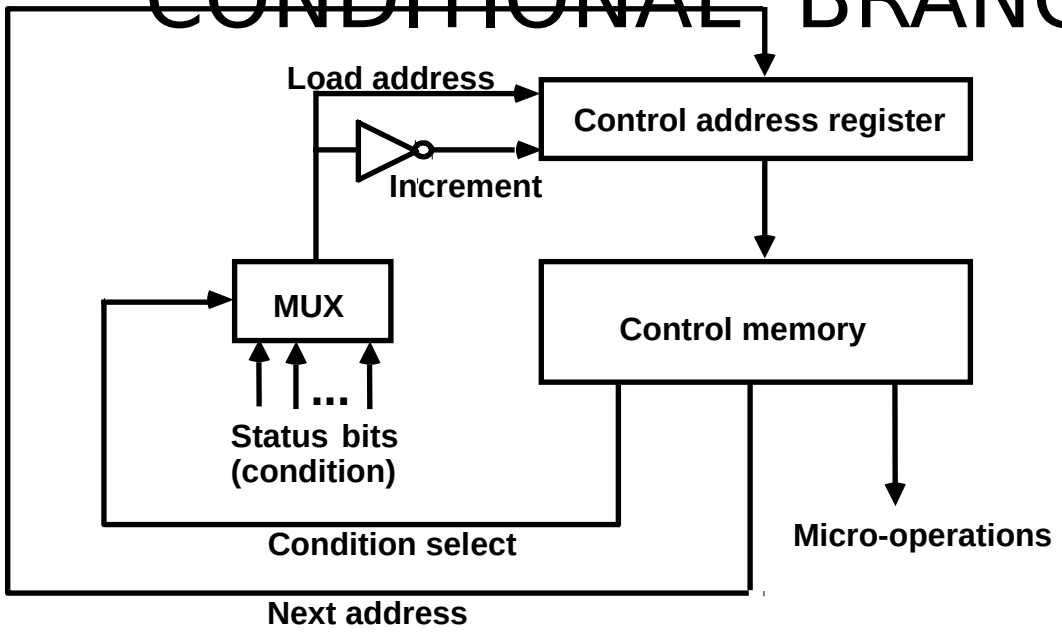
Overview

- 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.

CONDITIONAL BRANCH



Conditional Branch

If *Condition* is true, then *Branch* (address from the next address field of the current microinstruction) else *Fall Through*

Conditions to Test: O(overflow), N(negative), Z(zero), C(carry), etc.

Unconditional Branch

Fixing the value of one status bit at the input of the multiplexer to 1

Overview

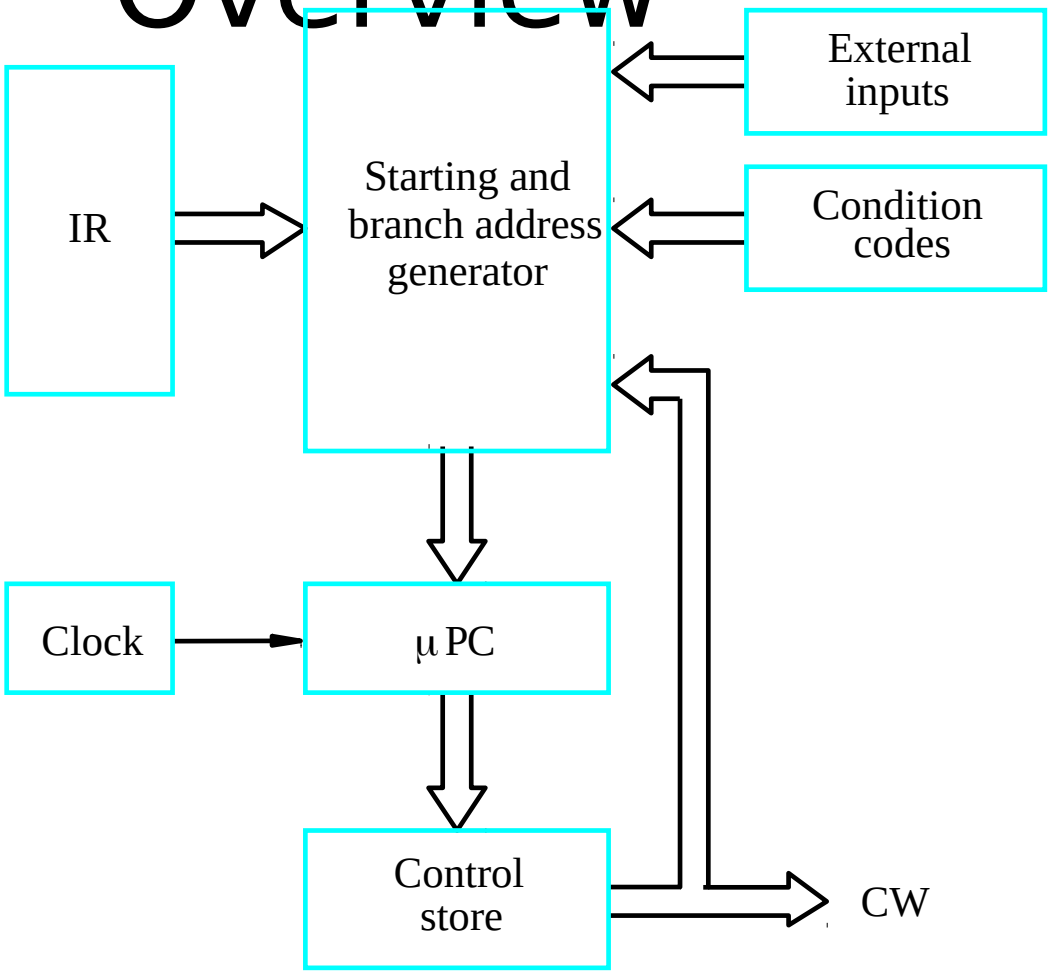
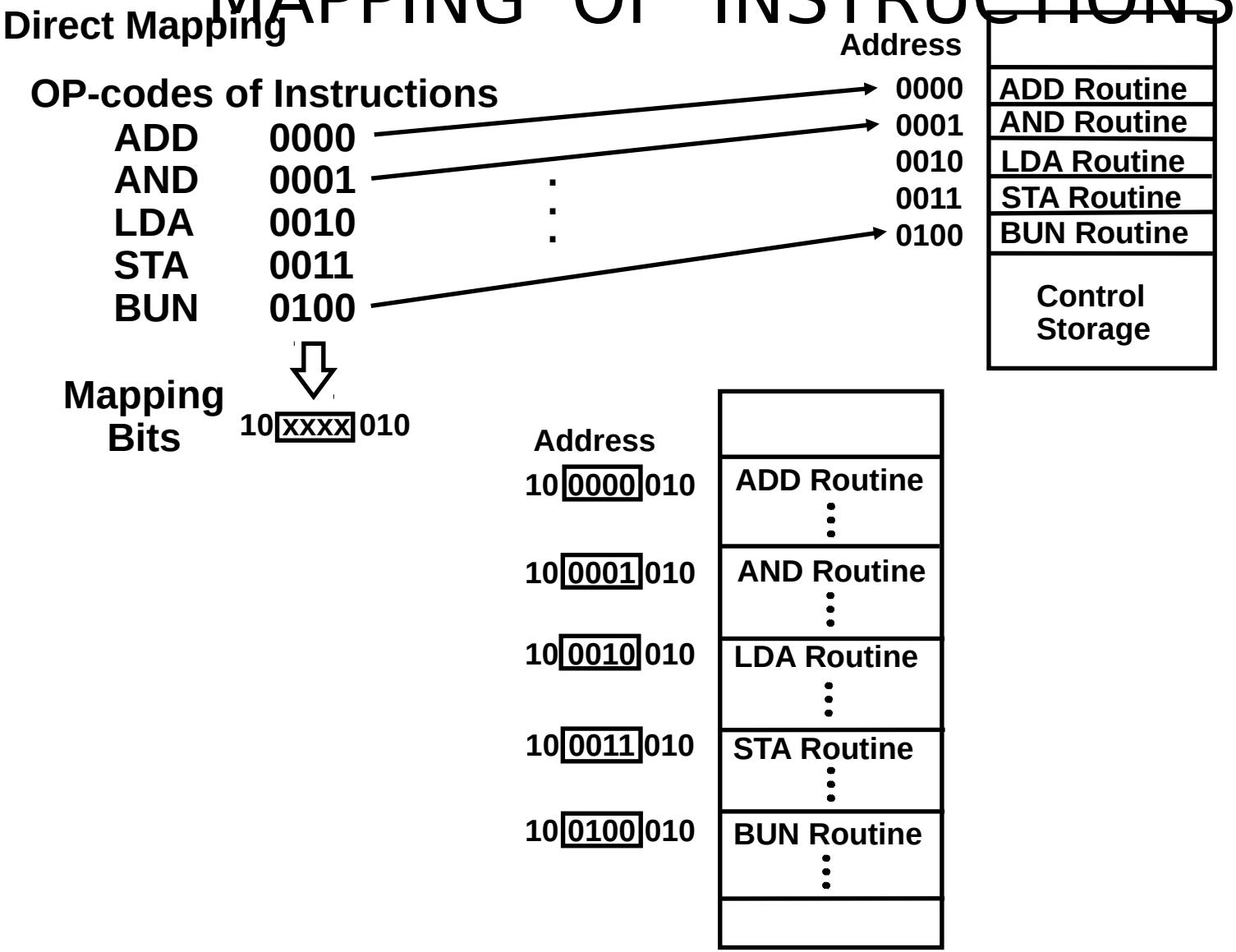


Figure 7.18. Organization of the control unit to allow conditional branching in the microprogram.

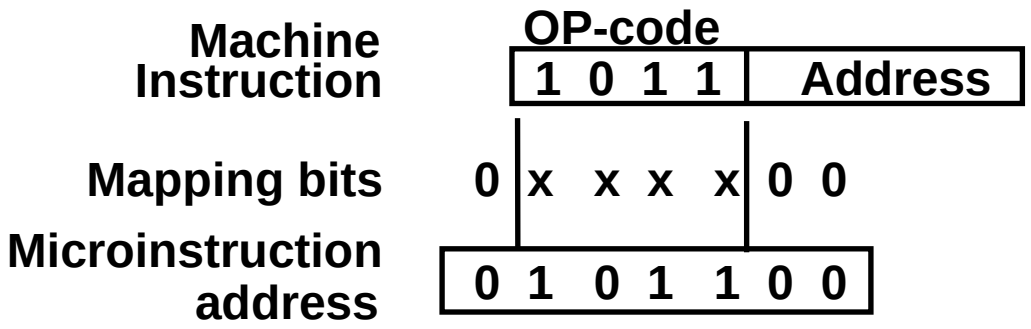
MAPPING OF INSTRUCTIONS

Direct Mapping

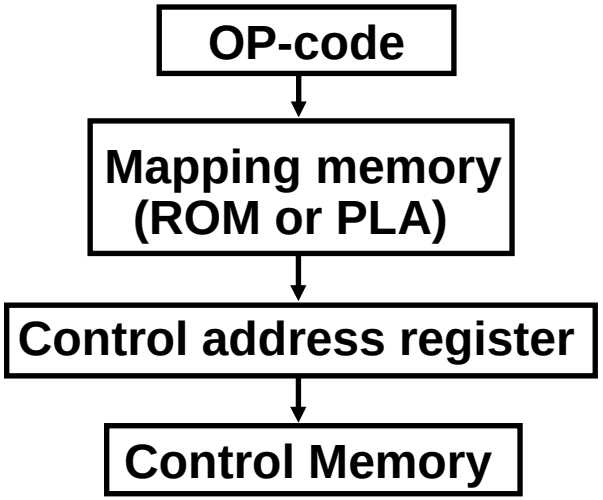


MAPPING OF INSTRUCTIONS TO MICROROUTINES

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

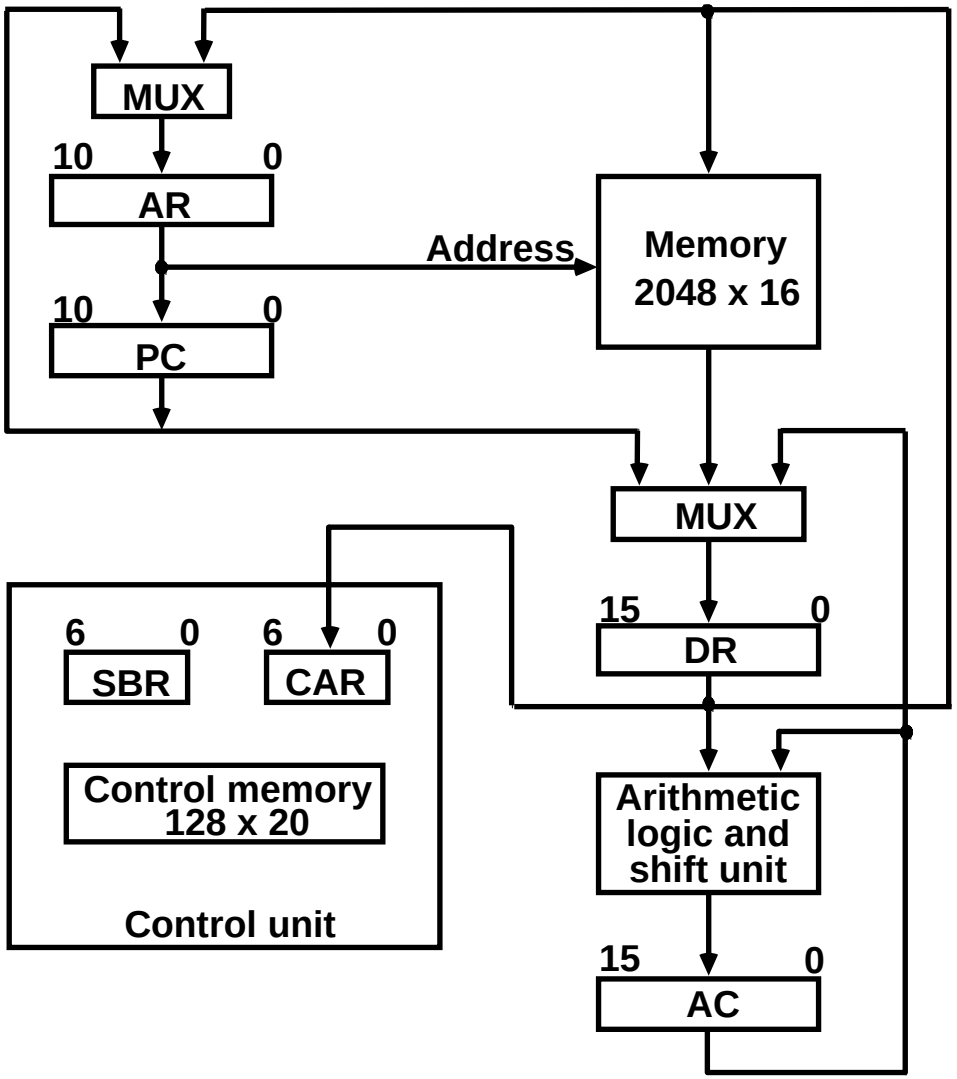


Mapping function implemented by ROM or PLA



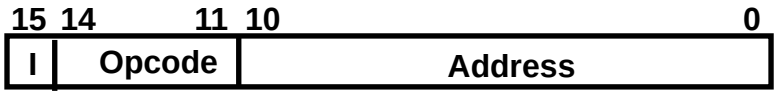
MICROPROGRAM EXAMPLE

Computer Configuration



MACHINE INSTRUCTION FORMAT

Machine instruction format

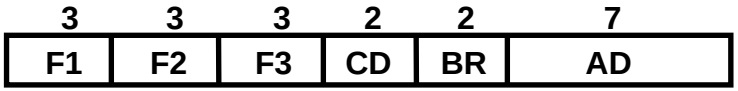


Sample machine instructions

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

SYMBOLIC MICROPROGRAM

- **Control Storage: 128 20-bit words**
- **The first 64 words: Routines for the 16 machine instructions**
- **The last 64 words: Used for other purpose (e.g., fetch routine and other subroutines)**
- **Mapping: OP-code XXXX into 0XXXX00, the first address for the 16 routines are 0(0 0000 00), 4(0 0001 00), 8, 12, 16, 20, ..., 60**

Partial Symbolic Microprogram

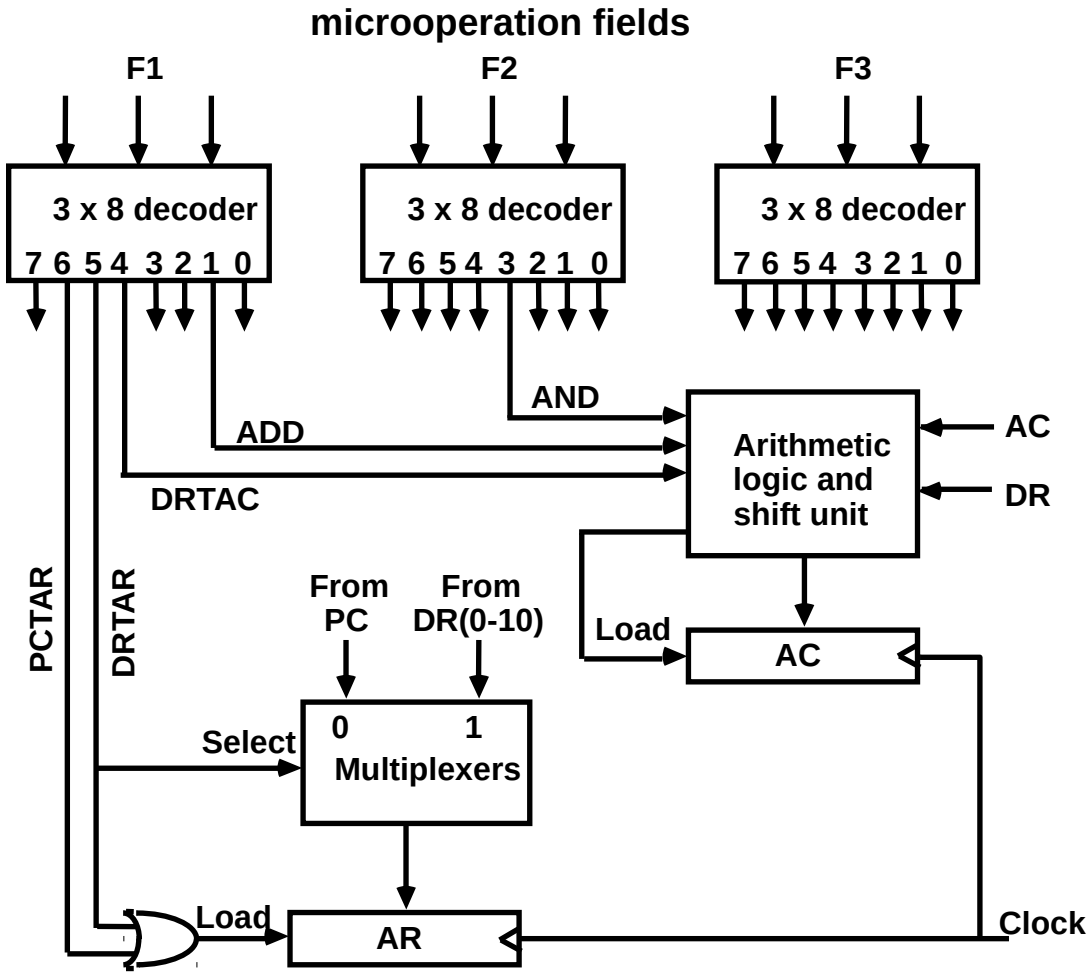
Label	Microops	CD	BR	AD
	ORG 0			
ADD:	NOP	I	CALL	INDRCT
	READ	U	JMP	NEXT
	ADD	U	JMP	FETCH
	ORG 4			
BRANCH:	NOP	S	JMP	OVER
	NOP	U	JMP	FETCH
OVER:	NOP	I	CALL	INDRCT
	ARTPC	U	JMP	FETCH
	ORG 8			
STORE:	NOP	I	CALL	INDRCT
	ACTDR	U	JMP	NEXT
	WRITE	U	JMP	FETCH
	ORG 12			
EXCHANGE:	NOP	I	CALL	INDRCT
	READ	U	JMP	NEXT
	ACTDR, DRTAC	U	JMP	NEXT
	WRITE	U	JMP	FETCH
	ORG 64			
FETCH:	PCTAR	U	JMP	NEXT
	READ, INCPC	U	JMP	NEXT
	DRTAR	U	MAP	
INDRCT:	READ	U	JMP	NEXT
	DRTAR	U	RET	

BINARY MICROPROGRAM

Micro Routine	Address Decimal	Address Binary	Binary Microinstruction F1	F2	F3	CD	BR	AD
ADD	0	0000000	000	000	01	01	10	000011
	1	0000001	000	100	000	00	00	0000010
	2	0000010	001	000	000	00	00	1000000
	3	0000011	000	000	000	00	00	1000000
BRANCH	4	0000100	000	000	000	10	00	0000110
	5	0000101	000	000	000	00	00	1000000
	6	0000110	000	000	000	01	01	1000011
	7	0000111	000	000	110	00	00	1000000
STORE	8	0001000	000	000	000	01	01	1000011
	9	0001001	000	101	000	00	00	0001010
	10	0001010	111	000	000	00	00	1000000
	11	0001011	000	000	000	00	00	1000000
EXCHANGE	12	0001100	000	000	000	01	01	1000011
	13	0001101	001	000	000	00	00	0001110
	14	0001110	100	101	000	00	00	0001111
	15	0001111	010	000	000	00	00	1000000
FETCH	64	1000000	110	000	000	00	00	1000001

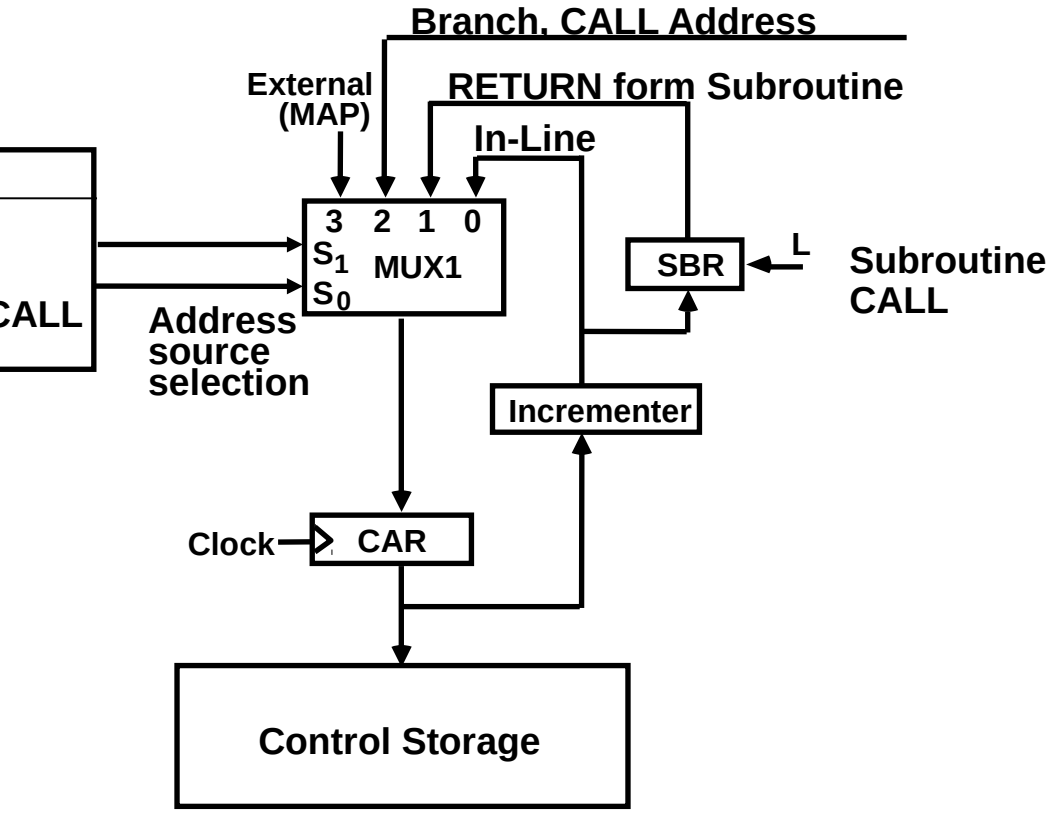
This microprogram can be implemented using ROM

DESIGN OF CONTROL UNIT
- DECODING ALU CONTROL INFORMATION -



MICROPROGRAM SEQUENCER
- NEXT MICROINSTRUCTION ADDRESS LOGIC -

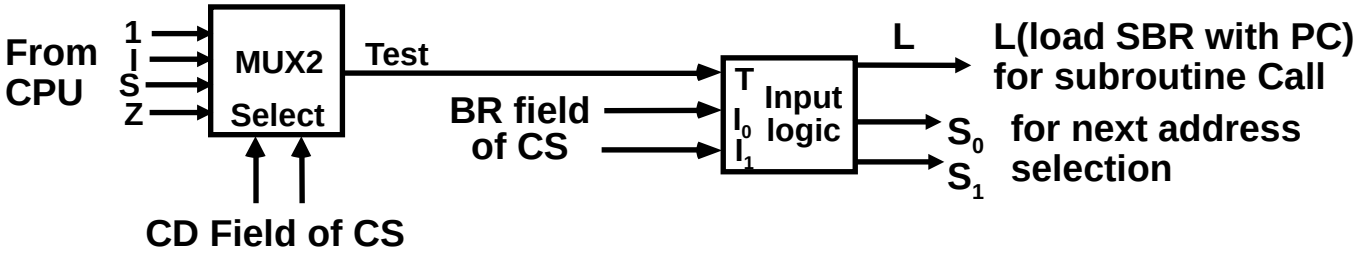
S_1S_0	Address Source
00	CAR + 1, In-Line
01	SBR RETURN
10	CS(AD), Branch or CALL
11	MAP



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**

MICROPROGRAM SEQUENCER
- CONDITION AND BRANCH CONTROL -



Input Logic

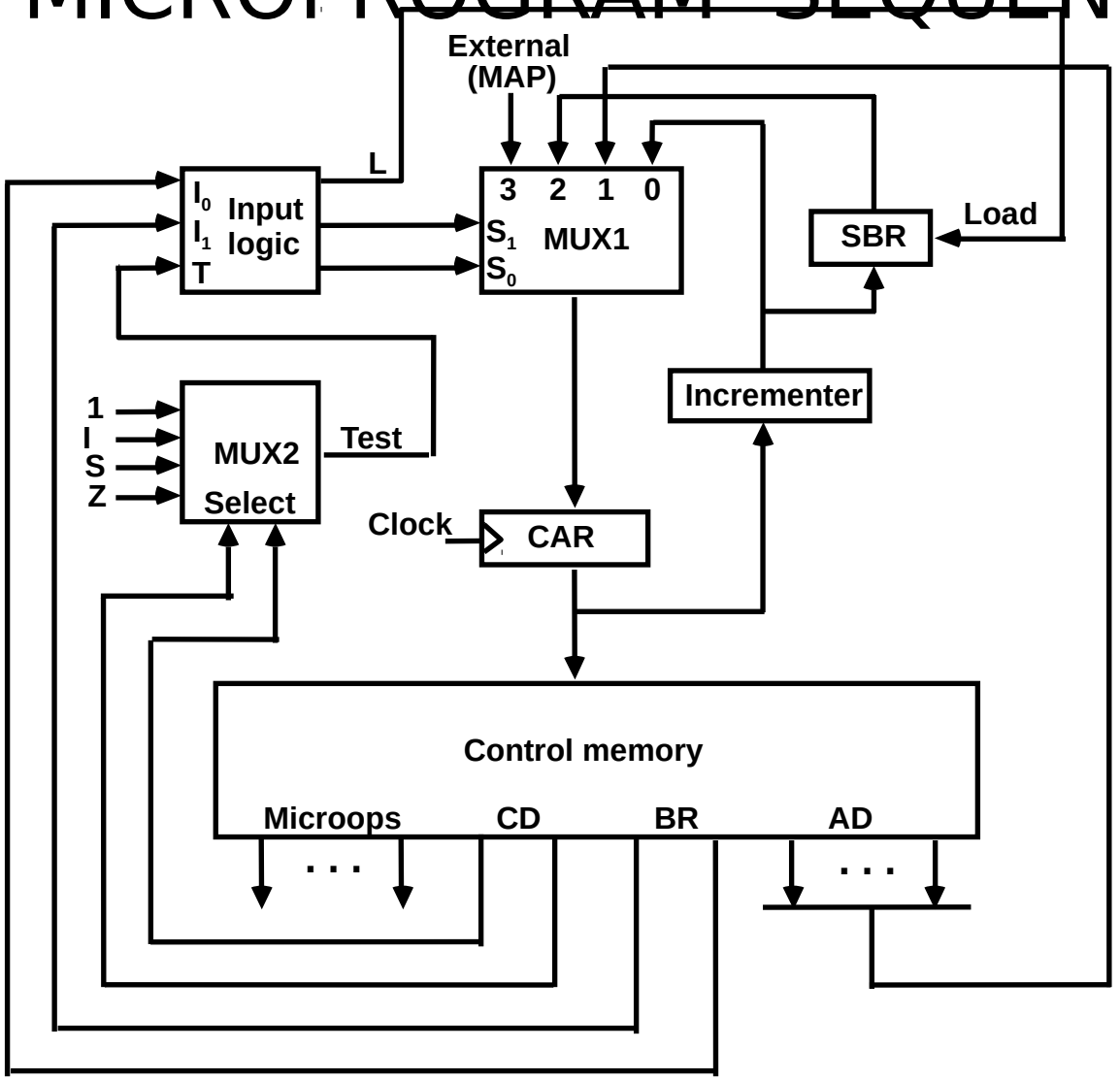
I_0I_1T	Meaning	Source of Address	S_1S_0	L
000	In-Line	CAR+1	00	0
001	JMP	CS(AD)	10	0
010	In-Line	CAR+1	00	0
011	CALL	CS(AD) and SBR <- CAR+1	10	1
10x	RET	SBR	01	0
11x	MAP	DR(11-14)	11	0

$S_0 = I_0$

$S_1 = I_0I_1 + I_0'T$

$L = I_0'I_1T$

MICROPROGRAM SEQUENCER



MICROINSTRUCTION FORMAT

Information in a Microinstruction

- **Control Information**
- **Sequencing Information**
- **Constant**

  Information which is useful when feeding into the system

These information needs to be organized in some way for

- **Efficient use of the microinstruction bits**
- **Fast decoding**

Field Encoding

- **Encoding the microinstruction bits**
- **Encoding slows down the execution speed due to the decoding delay**
- **Encoding also reduces the flexibility due to the decoding hardware**

HORIZONTAL AND VERTICAL MICROINSTRUCTION FORMAT

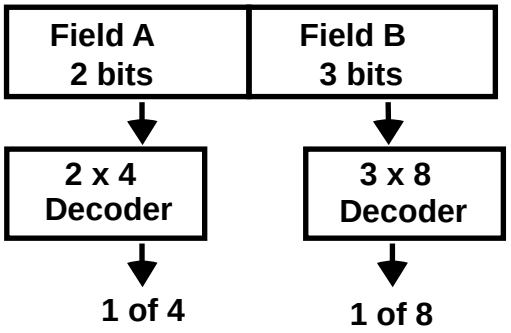
Horizontal Microinstructions

- **Each bit directly controls each micro-operation or each control point**
- ***Horizontal* implies a long microinstruction word**
- **Advantages: Can control a variety of components operating in parallel.**
  - **--> Advantage of efficient hardware utilization**
- **Disadvantages: Control word bits are not fully utilized**
  - **--> CS becomes large --> Costly**

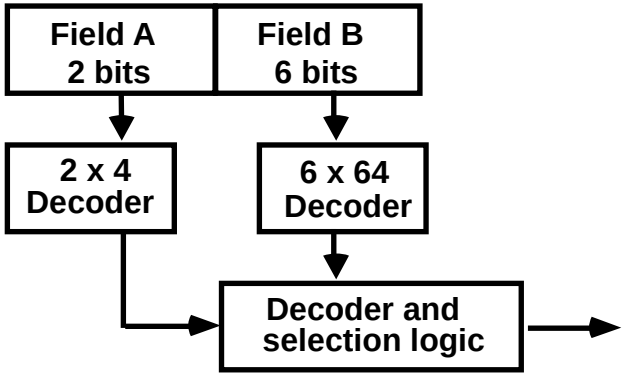
Vertical Microinstructions

- **A microinstruction format that is not horizontal**
- ***Vertical* implies a short microinstruction word**
- **Encoded Microinstruction fields**
- **--> Needs decoding circuits for one or two levels of decoding**

One-level decoding



Two-level decoding



NANOSTORAGE AND NANOINSTRUCTION

The decoder circuits in a vertical microprogram storage organization can be replaced by a ROM

=> Two levels of control storage

First level - *Control Storage*

Second level - *Nano Storage*

Two-level microprogram

First level

-*Vertical* format Microprogram

Second level

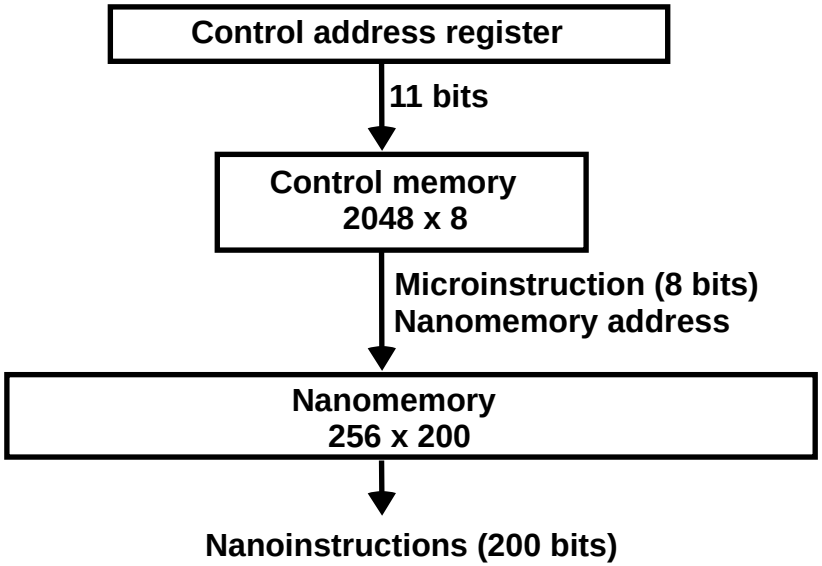
-*Horizontal* format Nanoprogram

- Interprets the microinstruction fields, thus converts a vertical microinstruction format into a horizontal nanoinstruction format.

Usually, the microprogram consists of a large number of short microinstructions, while the nanoprogram contains fewer words with longer nanoinstructions.

TWO-LEVEL MICROPROGRAMMING - EXAMPLE

*** Microprogram: 2048 microinstructions of 200 bits each**
*** With 1-Level Control Storage: 2048 x 200 = 409,600 bits**
*** Assumption:**
256 distinct microinstructions among 2048
*** With 2-Level Control Storage:**
Nano Storage: 256 x 200 bits to store 256 distinct nanoinstructions
Control storage: 2048 x 8 bits
To address 256 nano storage locations 8 bits are needed
*** Total 1-Level control storage: 409,600 bits**
Total 2-Level control storage: 67,584 bits (256 x 200 + 2048 x 8)



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 (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_{in}	001: MAR_{in}	0001: Sub	01: Read
0010: MDR_{out}	010: IR_{in}	010: MDR_{in}	⋮	10: Write
0011: Z_{out}	011: Z_{in}	011: $TEMP_{in}$	1111: XOR	
0100: $R0_{out}$	100: $R0_{in}$	100: Y_{in}	16 ALU functions	
0101: $R1_{out}$	101: $R1_{in}$			
0110: $R2_{out}$	110: $R2_{in}$			
0111: $R3_{out}$	111: $R3_{in}$			
1010: $TEMP_{out}$				
1011: $Offset_{out}$				

F6 (1 bit)	F7 (1 bit)	F8 (1 bit)
0: SelectY 1: Select4	0: No action 1: WMFC	0: Continue 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

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

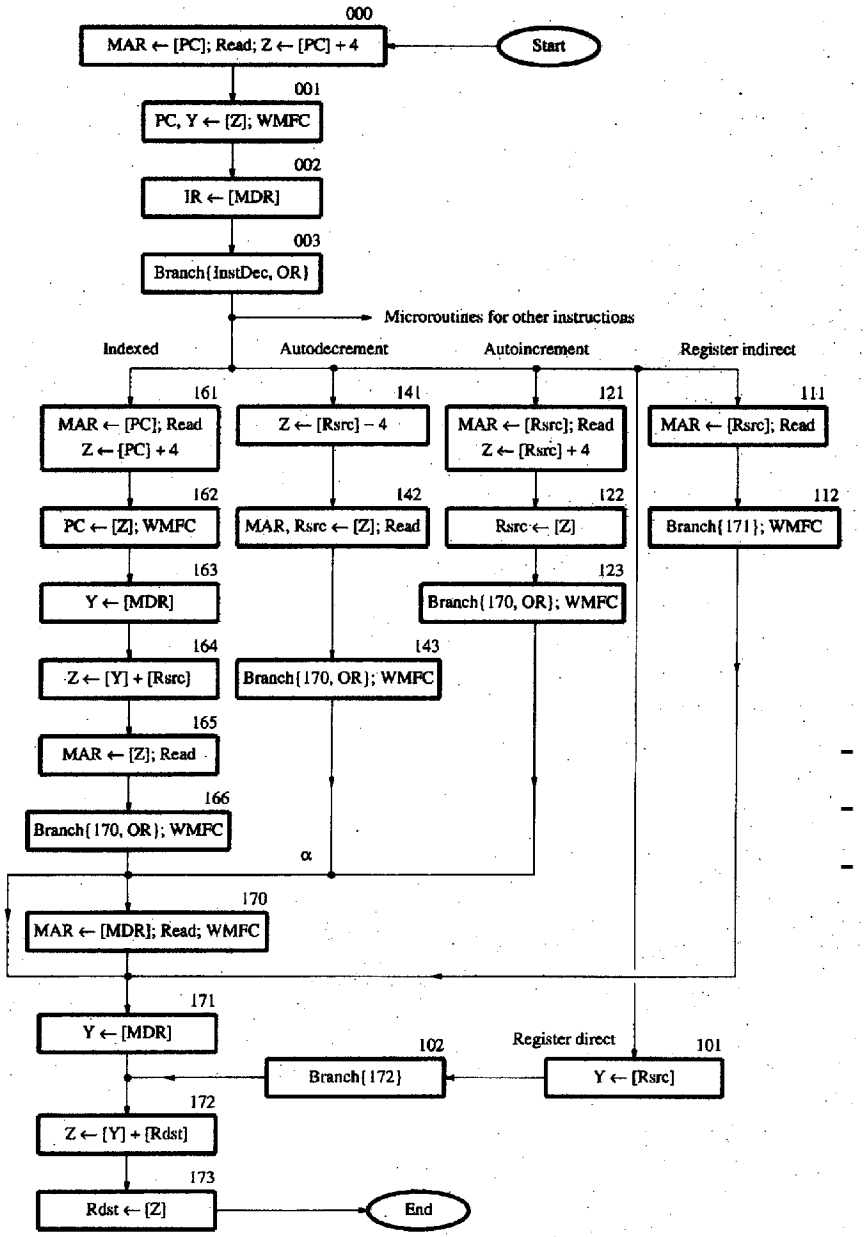
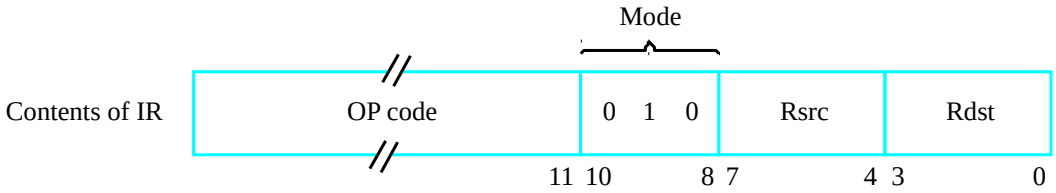


Figure 7.20. Flowchart of a microprogram for the Add src,Rdst instruction.

- Bit-ORing
- Wide-Branch Addressing
- WMFC



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

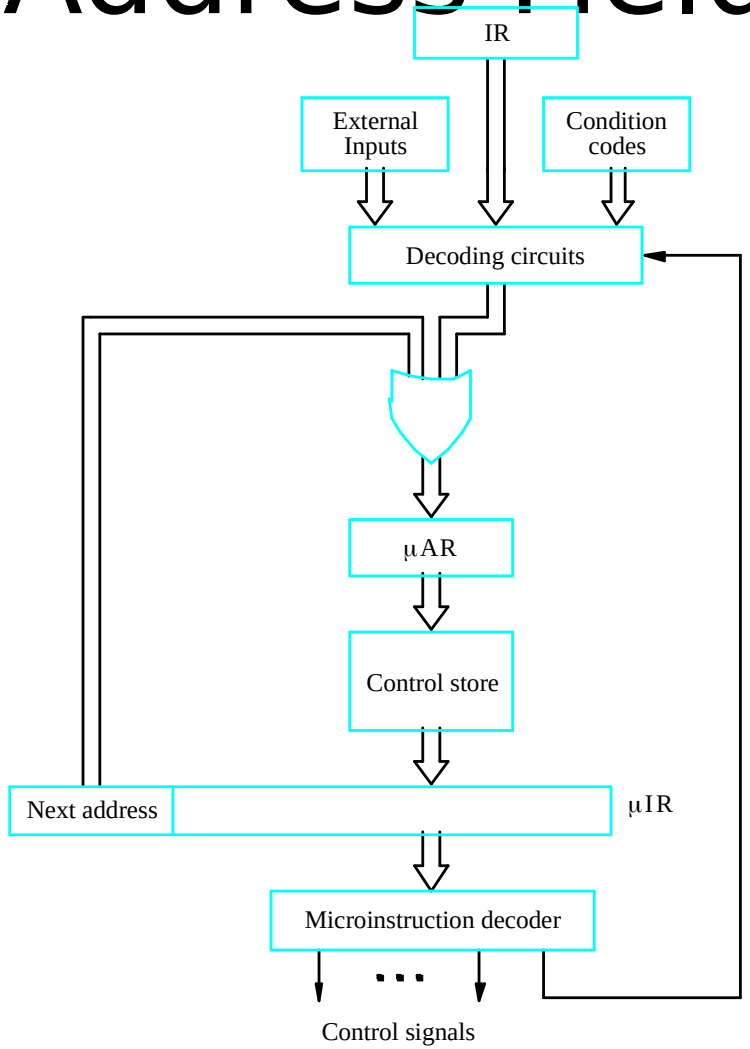


Figure 7.22. Microinstruction-sequencing organization.

Microinstruction

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 (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 (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.

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

Control Sequencer

- Sends out control words (16 bit in example)
  - One during each T state or clock cycle
- Each is a instruction telling computer what to do
- These instructions (of 16 bits 1 s and 0s) are called microinstructions

Control Memory(ROM)

Fetch

LDA

ADD

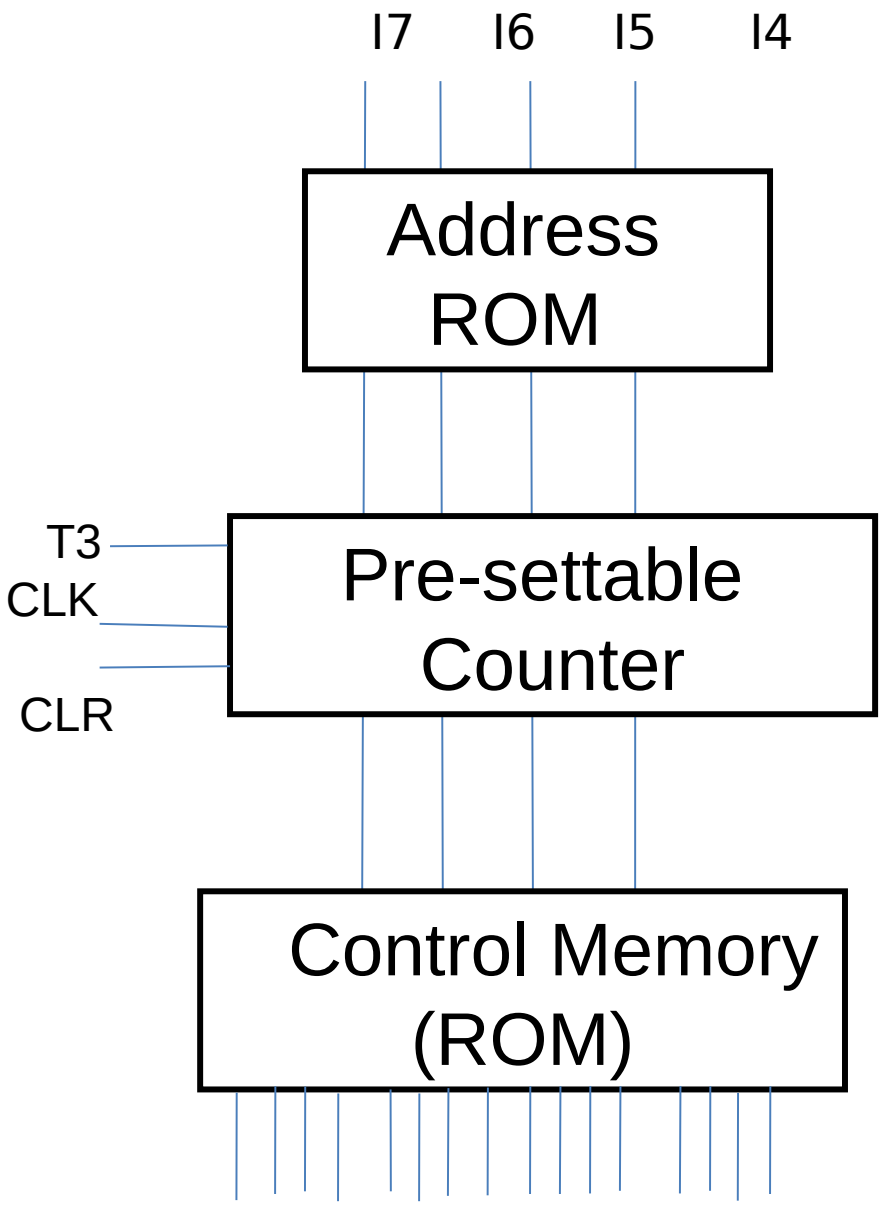
Address	Control Word
0 H	.....
1 H	.....
2H	.....
3H	.....
4H	.....
5H	....
6H	.
7H	.
8H	.

Address ROM

Op Code	Starting Address
0000 (LDA)	0011
0001 (ADD)	0110
0010 (SUB)	1001
	.....
	....
	.

When counter reset by CLR ,Counter output is 0000 at T1,0001 at T2, 0010 at T3– always for instruction fetch.

Opcode say ADD has been fetched. I7 ,I6 I5 , I4 bits are 0001.Address ROM produces 0110-- is the input to presettable counter,when T3 is high.Counter is preset to 0110 at T4 state…..at T6 it is 1000.State T1 again clears and starts address 0000 Fetch

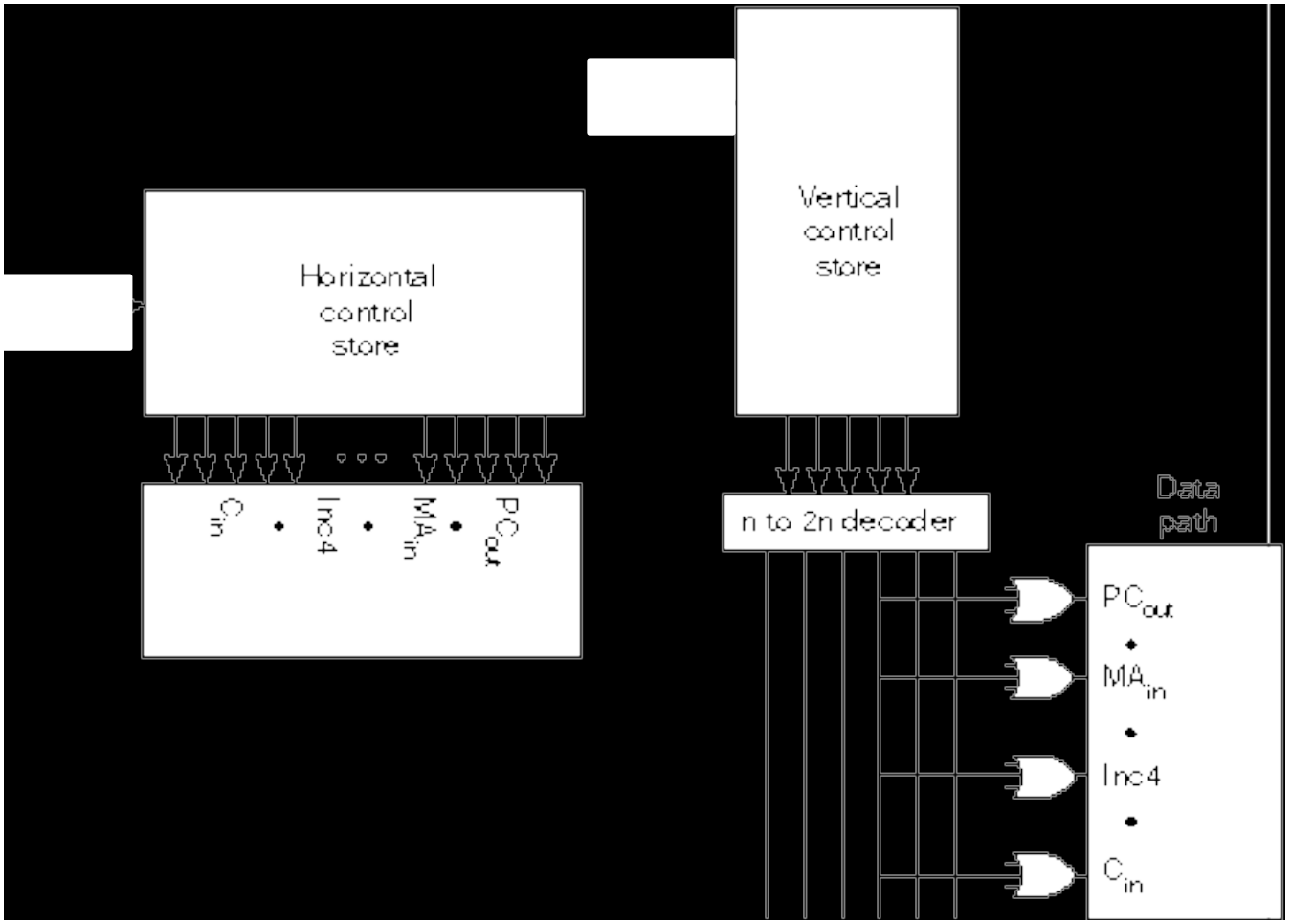


Control Word (16 bits in the example

Horizontal Versus Vertical Microcode Schemes

- In **horizontal** microcode, each control signal is represented by a bit in the μinstruction(Microinstruction)
  - Fewer control store words of more bits per word
- In **vertical** microcode, a set of control signals is represented by a shorter code
  - Vertical μcode only allows RTs(register transfers) in a step for which there is a vertical μinstruction code
  - Thus vertical μcode may take more control

Completely Horizontal and Vertical Microcoding



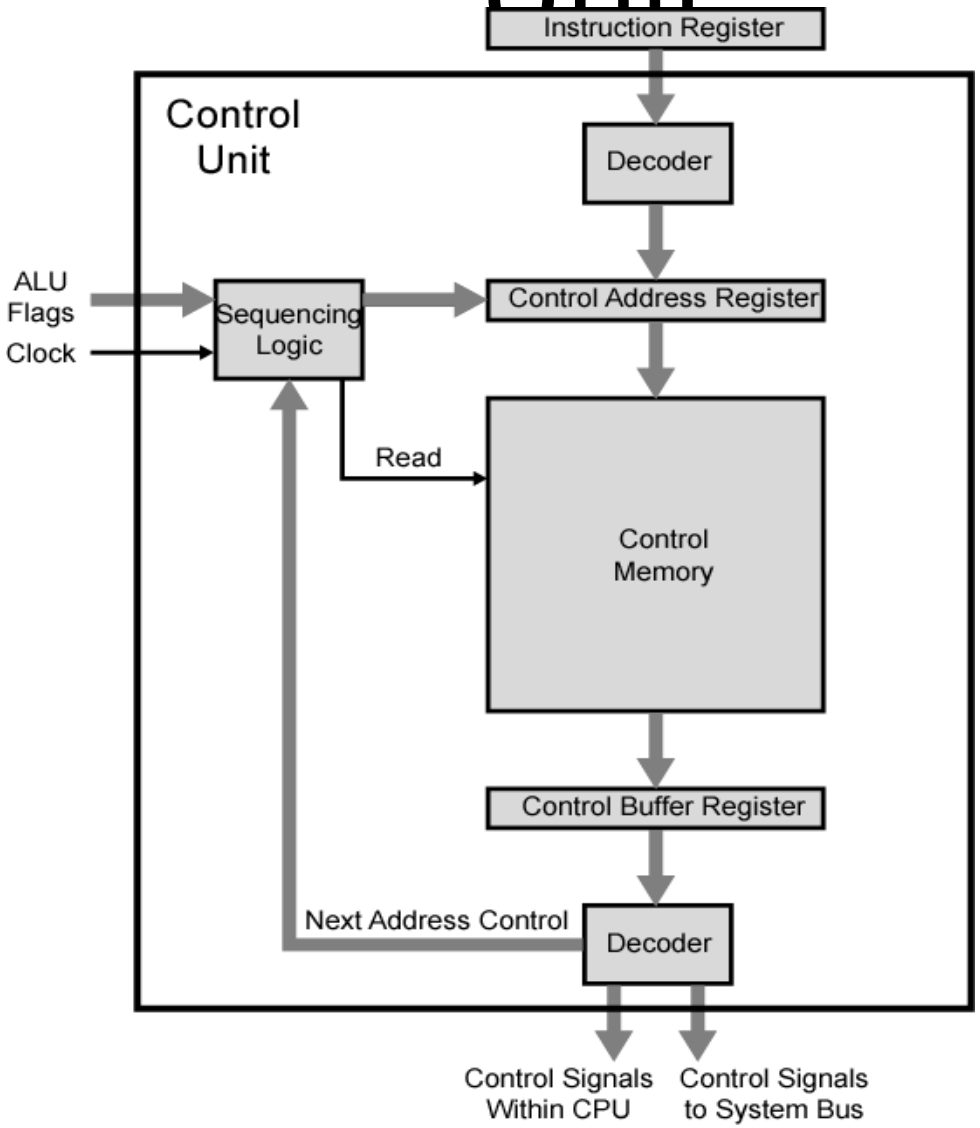
Horizontal Micro-programming

- Wide **control memory word**
- High degree of parallel operations possible
- Little encoding of control information
- Fast

Vertical Micro-programming

- Width can be much narrower
- Control signals encoded into function codes – need to be decoded
- More complex, more complicated to program, less flexibility
- More difficult to modify
- Slower

Microprogrammed Control Unit



Next Address Decision

- Depending on ALU flags and control buffer register:
  - Get next instruction
    - Add 1 to control address register
  - Jump to new routine based on jump microinstruction
    - Load address field of control buffer register into control address register
  - Jump to machine instruction routine
    - Load control address register based on

Advantages and Disadvantages of Microprogramming

Advantage:

- Simplifies design of control unit
  - Cheaper
  - Less error-prone
  - Easier to modify

Disadvantage:

- Slower

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	

An example of microinstructions fc