PROGRAMMING THE BASIC COMPUTER

Introduction

Machine Language

Assembly Language

Assembler

Program Loops

Programming Arithmetic and Logic Operations

Subroutines

Input-Output Programming

INTRODUCTION

Those concerned with computer architecture should have a knowledge of both hardware and software because the two branches influence each other.

Instruction Set of the *Basic Computer*

Symbol	*Hexa code*	*Description*
AND	0 or 8	AND M to AC
ADD	1 or 9	Add M to AC, carry to E
LDA	2 or A	Load AC from M
STA	3 or B	Store AC in M
BUN	4 or C	Branch unconditionally to m
BSA	5 or D	Save return address in m and branch to m+1
ISZ	6 or E	Increment M and skip if zero
CLA	7800	Clear AC
CLE	7400	Clear E
CMA	7200	Complement AC
CME	7100	Complement E
CIR	7080	Circulate right E and AC
CIL	7040	Circulate left E and AC
INC	7020	Increment AC, carry to E
SPA	7010	Skip if AC is positive
SNA	7008	Skip if AC is negative
SZA	7004	Skip if AC is zero
SZE	7002	Skip if E is zero
HLT	7001	Halt computer
INP	F800	Input information and clear flag
OUT	F400	Output information and clear flag
SKI	F200	Skip if input flag is on
SKO	F100	Skip if output flag is on
ION	F080	Turn interrupt on
IOF	F040	Turn interrupt off

m: effective address
M: memory word (operand) found at m

MACHINE LANGUAGE

- **Program**
  A list of instructions or statements for directing the computer to perform a required data processing task

- **Various types of programming languages**
  - Hierarchy of programming languages

    - **Machine-language**
      - Binary code
      - Octal or hexadecimal code

    - **Assembly-language (Assembler)**
      - Symbolic code

    - **High-level language (Compiler)**

COMPARISON OF PROGRAMMING LANGUAGES

• Binary Program to Add Two Numbers

Location	Instruction Code
0	0010 0000 0000 0100
1	0001 0000 0000 0101
10	0011 0000 0000 0110
11	0111 0000 0000 0001
100	0000 0000 0101 0011
101	1111 1111 1110 1001
110	0000 0000 0000 0000

• Hexa program

Location	Instruction
000	2004
001	1005
002	3006
003	7001
004	0053
005	FFE9
006	0000

• Program with Symbolic OP-Code

Location	Instruction		Comments
000	LDA	004	Load 1st operand into AC
001	ADD	005	Add 2nd operand to AC
002	STA	006	Store sum in location 006
003	HLT		Halt computer
004	0053		1st operand
005	FFE9		2nd operand (negative)
006	0000		Store sum here

• Assembly-Language Program

```
      ORG   0      /Origin of program is location 0
      LDA   A      /Load  operand from location A
      ADD   B      /Add  operand from location B
      STA   C      /Store sum in location C
      HLT          /Halt computer
A,    DEC   83     /Decimal operand
B,    DEC   -23    /Decimal operand
C,    DEC   0      /Sum stored in location C
      END          /End of symbolic program
```

• Fortran Program

```
INTEGER  A, B, C
DATA  A,83 / B,-23
C = A + B
END
```

ASSEMBLY LANGUAGE

Syntax of the BC assembly language

Each line is arranged in three columns called fields

***Label* field**

- May be empty or may specify a symbolic address consists of up to 3 characters

- Terminated by a comma

***Instruction* field**

- Specifies a machine or a pseudo instruction

- May specify one of

*** Memory reference instr. (MRI)**

MRI consists of two or three symbols separated by spaces.

ADD OPR (direct address MRI)

ADD PTR I (indirect address MRI)

*** Register reference or input-output instr.**

Non-MRI does not have an address part

*** Pseudo instr. with or without an operand**

Symbolic address used in the instruction field must be defined somewhere as a label

***Comment* field**

- May be empty or may include a comment

PSEUDO-INSTRUCTIONS

ORG N
Hexadecimal number N is the memory loc.
for the instruction or operand listed in the following line

END
Denotes the end of symbolic program

DEC N
Signed decimal number N to be converted to the binary

HEX N
Hexadecimal number N to be converted to the binary

Example: Assembly language program to subtract two numbers

```
        ORG  100        / Origin of program is location 100
        LDA  SUB        / Load subtrahend to AC
        CMA             / Complement AC
        INC             / Increment AC
        ADD  MIN        / Add minuend to AC
        STA  DIF        / Store difference
        HLT             / Halt computer
MIN,    DEC  83         / Minuend
SUB,    DEC  -23        / Subtrahend
DIF,    HEX  0          / Difference stored here
        END             / End of symbolic program
```

TRANSLATION TO BINARY

Hexadecimal Code		*Symbolic Program*	
Location	*Content*		
			ORG 100
100	2107		LDA SUB
101	7200		CMA
102	7020		INC
103	1106		ADD MIN
104	3108		STA DIF
105	7001		HLT
106	0053	MIN,	DEC 83
107	FFE9	SUB,	DEC -23
108	0000	DIF,	HEX 0
			END

ASSEMBLER - FIRST PASS -

Assembler

Source Program - Symbolic Assembly Language Program

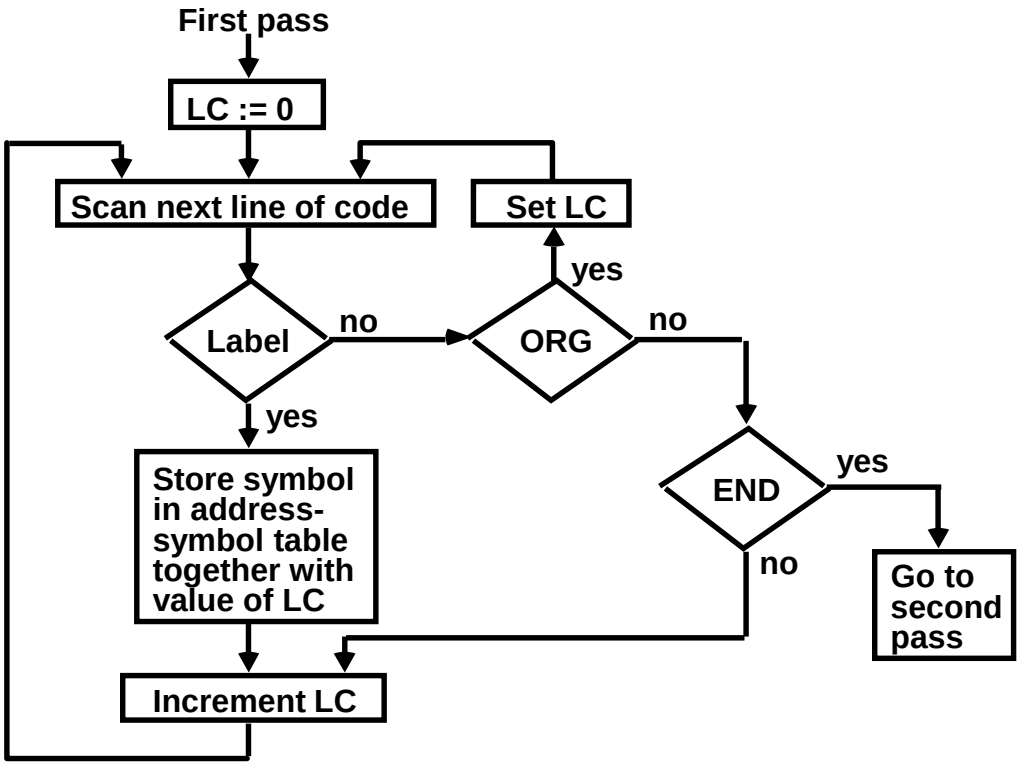
Object Program - Binary Machine Language Program

Two pass assembler

1st pass: generates a table that correlates all user defined (address) symbols with their binary equivalent value

2nd pass: binary translation

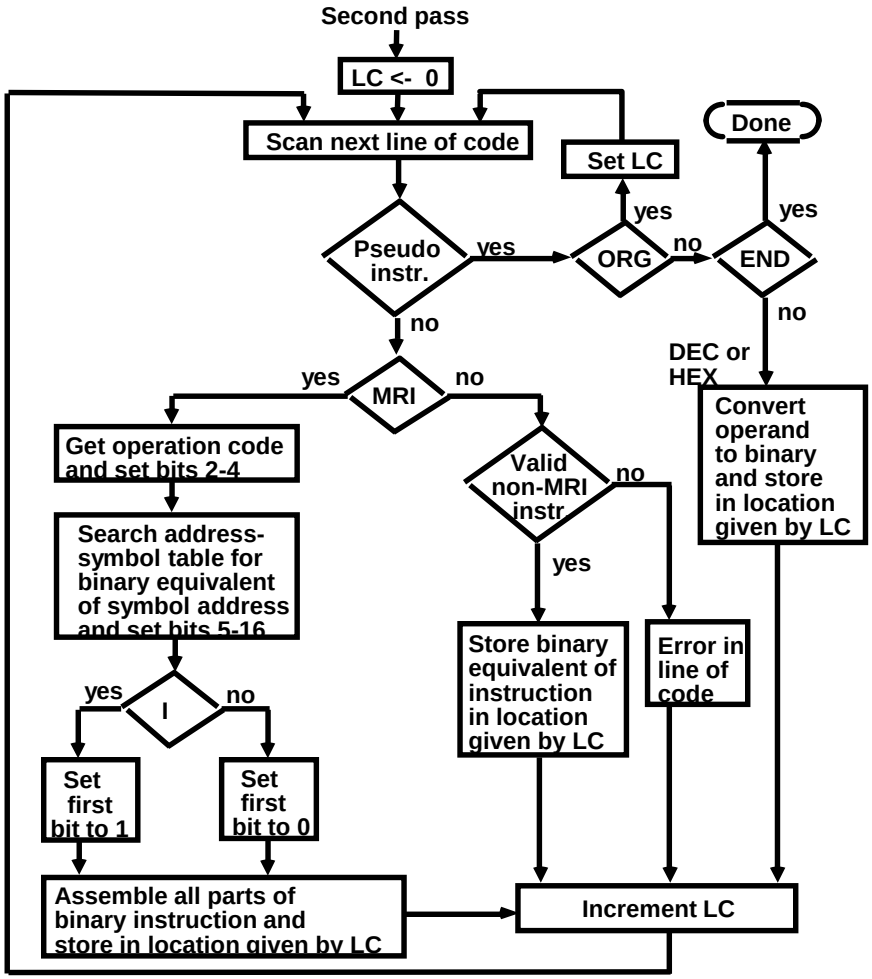
First pass



ASSEMBLER - SECOND PASS -

Second Pass

Machine instructions are translated by means of table-lookup procedures;
(1. Pseudo-Instruction Table, 2. MRI Table, 3. Non-MRI Table
4. Address Symbol Table)



PROGRAM LOOPS

Loop: **A sequence of instructions that are executed many times, each with a different set of data**

Fortran program to add 100 numbers:

```
      DIMENSION  A(100)
      INTEGER  SUM,  A
      SUM = 0
      DO  3  J = 1,  100
3     SUM = SUM + A(J)
```

Assembly-language program to add 100 numbers:

```
          ORG  100          / Origin of program is HEX 100
          LDA  ADS          / Load first address of operand
          STA  PTR          / Store in pointer
          LDA  NBR          / Load -100
          STA  CTR          / Store in counter
          CLA               / Clear AC
LOP,      ADD  PTR  I       / Add an operand to AC
          ISZ  PTR          / Increment pointer
          ISZ  CTR          / Increment counter
          BUN  LOP          / Repeat loop again
          STA  SUM          / Store sum
          HLT               / Halt
ADS,      HEX  150          / First address of operands
PTR,      HEX  0            / Reserved for a pointer
NBR,      DEC  -100         / Initial value for a counter
CTR,      HEX  0            / Reserved for a counter
SUM,      HEX  0            / Sum is stored here
          ORG  150          / Origin of operands is HEX 150
          DEC  75           / First operand
             :
             :
          DEC  23           / Last operand
          END               / End of symbolic program
```

PROGRAMMING ARITHMETIC AND LOGIC OPERATIONS

Implementation of Arithmetic and Logic Operations

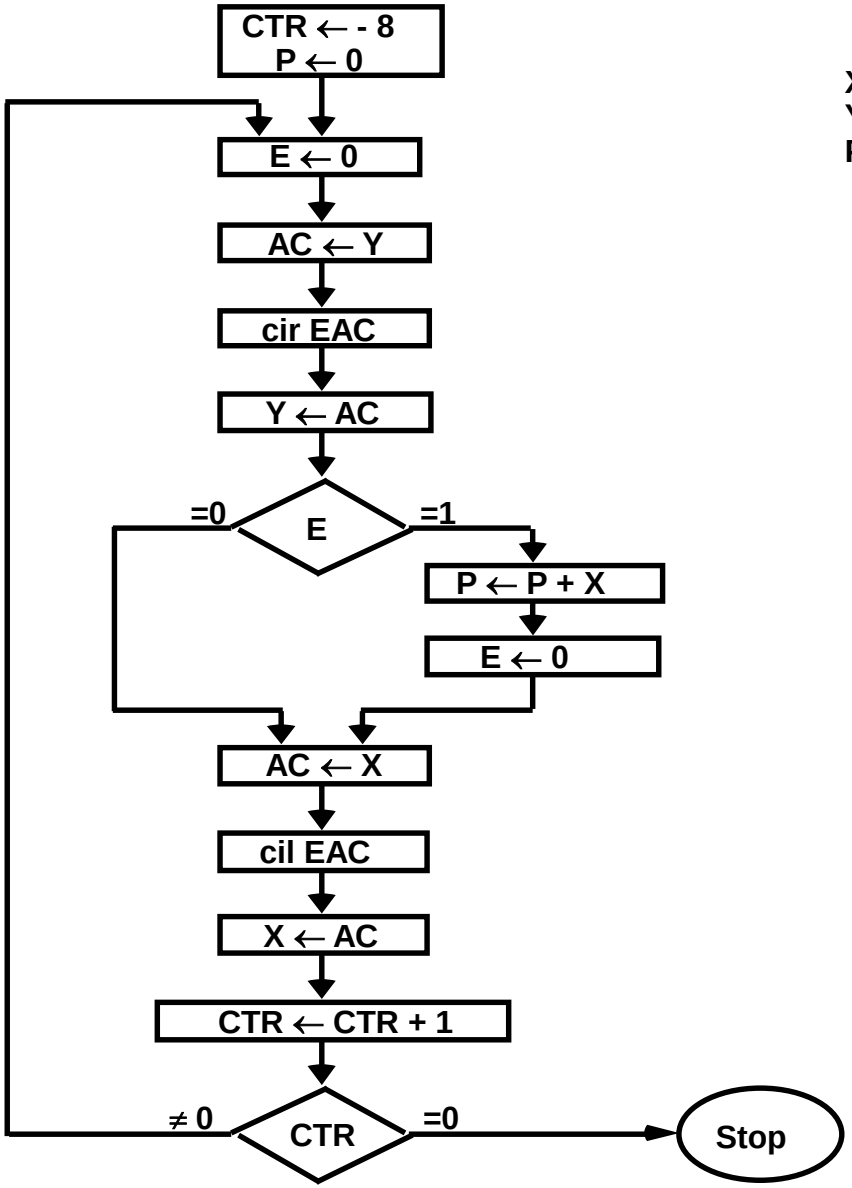
- Software Implementation
  - Implementation of an operation with a program using machine instruction set
  - Usually when the operation is not included in the instruction set

- Hardware Implementation
  - Implementation of an operation in a computer with one machine instruction

Software Implementation example:

* Multiplication
  - For simplicity, unsigned positive numbers
  - 8-bit numbers -> 16-bit product

FLOWCHART OF A PROGRAM - Multiplication -



X holds the multiplicand
Y holds the multiplier
P holds the product

Example with four significant digits

		P
X =	0000 1111	0000 0000
Y =	0000 1011	0000 1111
	0000 1111	0001 1110
	0001 1110	0000 0000
	0000 0000	0111 1000
	0111 1000	1010 0101
	1010 0101	

ASSEMBLY LANGUAGE PROGRAM - Multiplication -

```
        ORG  100
LOP,    CLE          / Clear E
        LDA  Y       / Load multiplier
        CIR          / Transfer multiplier bit to E
        STA  Y       / Store shifted multiplier
        SZE          / Check if bit is zero
        BUN  ONE     / Bit is one; goto ONE
        BUN  ZRO     / Bit is zero; goto ZRO
ONE,    LDA  X       / Load multiplicand
        ADD  P       / Add to partial product
        STA  P       / Store partial product
        CLE          / Clear E
ZRO,    LDA  X       / Load multiplicand
        CIL          / Shift left
        STA  X       / Store shifted multiplicand
        ISZ  CTR     / Increment counter
        BUN  LOP     / Counter not zero; repeat loop
        HLT          / Counter is zero; halt
CTR,    DEC  -8      / This location serves as a counter
X,      HEX  000F    / Multiplicand stored here
Y,      HEX  000B    / Multiplier stored here
P,      HEX  0       / Product formed here
        END
```

ASSEMBLY LANGUAGE PROGRAM
- Double Precision Addition -

```
LDA   AL      / Load A low
ADD   BL      / Add B low, carry in E
STA   CL      / Store in C low
CLA           / Clear AC
CIL           / Circulate to bring carry into AC(16)
ADD   AH      / Add A high and carry
ADD   BH      / Add B high
STA   CH      / Store in C high
HLT
```

ASSEMBLY LANGUAGE PROGRAM
- Logic and Shift Operations -

• **Logic operations**

- **BC instructions : AND, CMA, CLA**
- **Program for OR operation**

```
LDA   A      / Load 1st operand
CMA          / Complement to get A'
STA   TMP    / Store in a temporary location
LDA   B      / Load 2nd operand B
CMA          / Complement to get B'
AND   TMP    / AND with A' to get A' AND B'
CMA          / Complement again to get A OR B
```

• **Shift operations - BC has *Circular Shift* only**

- **Logical shift-right operation**

```
CLE
CIR
```

- **Logical shift-left operation**

```
CLE
CIL
```

- **Arithmetic right-shift operation**

```
CLE          / Clear E to 0
SPA          / Skip if AC is positive
CME          / AC is negative
CIR          / Circulate E and AC
```

SUBROUTINES

Subroutine

- A set of common instructions that can be used in a program many times.
- Subroutine *linkage* : a procedure for branching
to a subroutine and returning to the main program

Example

Loc.			
100		ORG 100	/ Main program
101		LDA X	/ Load X
102		BSA SH4	/ Branch to subroutine
103		STA X	/ Store shifted number
104		LDA Y	/ Load Y
105		BSA SH4	/ Branch to subroutine again
106		STA Y	/ Store shifted number
107		HLT	
108	X,	HEX 1234	
	Y,	HEX 4321	
			/ Subroutine to shift left 4 times
109	SH4,	HEX 0	/ Store return address here
10A		CIL	/ Circulate left once
10B		CIL	
10C		CIL	
10D		CIL	/ Circulate left fourth time
10E		AND MSK	/ Set AC(13-16) to zero
10F		BUN SH4 I	/ Return to main program
110	MSK,	HEX FFF0	/ Mask operand
		END	

SUBROUTINE PARAMETERS AND DATA LINKAGE

Linkage of Parameters and Data between the Main Program and a Subroutine
- **via Registers**
- **via Memory locations**
- **….**

Example: Subroutine performing *LOGICAL OR operation*; Need two parameters

Loc.			
200		ORG 200	
200		LDA X	/ Load 1st operand into AC
201		BSA OR	/ Branch to subroutine OR
202		HEX 3AF6	/ 2nd operand stored here
203		STA Y	/ Subroutine returns here
204		HLT	
205	X,	HEX 7B95	/ 1st operand stored here
206	Y,	HEX 0	/ Result stored here
207	OR,	HEX 0	/ Subroutine OR
208		CMA	/ Complement 1st operand
209		STA TMP	/ Store in temporary location
20A		LDA OR I	/ Load 2nd operand
20B		CMA	/ Complement 2nd operand
20C		AND TMP	/ AND complemented 1st operand
20D		CMA	/ Complement again to get OR
20E		ISZ OR	/ Increment return address
20F		BUN OR I	/ Return to main program
210	TMP,	HEX 0	/ Temporary storage
		END	

SUBROUTINE - Moving a Block of Data -

```
                        / Main program
        BSA  MVE        / Branch to subroutine
        HEX  100        / 1st address of source data
        HEX  200        / 1st address of destination data
        DEC  -16        / Number of items to move
        HLT
MVE,    HEX  0          / Subroutine MVE
        LDA  MVE  I     / Bring address of source
        STA  PT1        / Store in 1st pointer
        ISZ  MVE        / Increment return address
        LDA  MVE  I     / Bring address of destination
        STA  PT2        / Store in 2nd pointer
        ISZ  MVE        / Increment return address
        LDA  MVE  I     / Bring number of items
        STA  CTR        / Store in counter
        ISZ  MVE        / Increment return address
LOP,    LDA  PT1  I     / Load source item
        STA  PT2  I     / Store in destination
        ISZ  PT1        / Increment source pointer
        ISZ  PT2        / Increment destination pointer
        ISZ  CTR        / Increment counter
        BUN  LOP        / Repeat 16 times
        BUN  MVE  I     / Return to main program
PT1,    --
PT2,    --
CTR,    --
```

- Fortran subroutine

```
      SUBROUTINE  MVE (SOURCE, DEST, N)
      DIMENSION  SOURCE(N), DEST(N)
      DO  20  I = 1, N
20    DEST(I) = SOURCE(I)
      RETURN
      END
```

INPUT OUTPUT PROGRAM

Program to Input one Character(Byte)

```
CIF,    SKI          / Check input flag
        BUN  CIF     / Flag=0, branch to check again
        INP          / Flag=1, input character
        OUT          / Display to ensure correctness
        STA  CHR     / Store character
        HLT
CHR,    --           / Store character here
```

Program to Output a Character

```
        LDA  CHR     / Load character into AC
COF,    SKO          / Check output flag
        BUN  COF     / Flag=0, branch to check again
        OUT          / Flag=1, output character
        HLT
CHR,    HEX  0057    / Character is "W"
```

CHARACTER MANIPULATION

Subroutine to Input 2 Characters and pack into a word

```
IN2,    --          / Subroutine entry
FST,    SKI
        BUN  FST
        INP         / Input 1st character
        OUT
        BSA  SH4    / Logical Shift left 4 bits
        BSA  SH4    / 4 more bits
SCD,    SKI
        BUN  SCD
        INP         / Input 2nd character
        OUT
        BUN  IN2  I / Return
```

PROGRAM INTERRUPT

Tasks of Interrupt Service Routine

- Save the Status of CPU
  Contents of processor registers and Flags

- Identify the source of Interrupt
  Check which flag is set

- Service the device whose flag is set
  (Input Output Subroutine)

- Restore contents of processor registers and flags

- Turn the interrupt facility on

- Return to the running program
  Load PC of the interrupted program

INTERRUPT SERVICE ROUTINE

```
Loc.
  0     ZRO,    -           / Return address stored here
  1             BUN  SRV    / Branch to service routine
100             CLA         / Portion of running program
101             ION         / Turn on interrupt facility
102             LDA  X
103             ADD  Y      / Interrupt occurs here
104             STA  Z      / Program returns here after interrupt

                            / Interrupt service routine
200     SRV,    STA  SAC    / Store content of AC
                CIR         / Move E into AC(1)
                STA  SE     / Store content of E
                SKI         / Check input flag
                BUN  NXT    / Flag is off, check next flag
                INP         / Flag is on, input character
                OUT         / Print character
                STA  PT1  I / Store it in input buffer
                ISZ  PT1    / Increment input pointer
        NXT,    SKO         / Check output flag
                BUN  EXT    / Flag is off, exit
                LDA  PT2  I / Load character from output buffer
                OUT         / Output character
                ISZ  PT2    / Increment output pointer
        EXT,    LDA  SE     / Restore value of AC(1)
                CIL         / Shift it to E
                LDA  SAC    / Restore content of AC
                ION         / Turn interrupt on
                BUN  ZRO  I / Return to running program
        SAC,    -           / AC is stored here
        SE,     -           / E is stored here
        PT1,    -           / Pointer of input buffer
        PT2,    -           / Pointer of output buffer
```

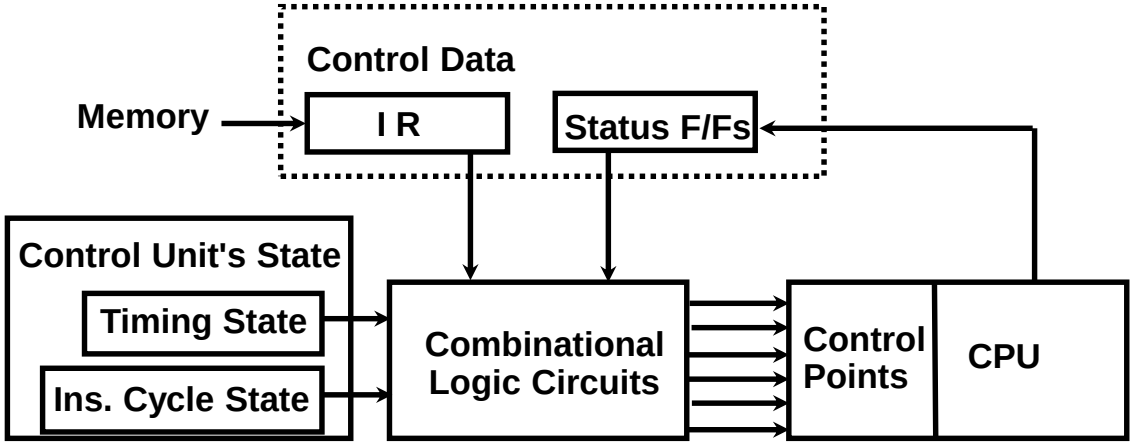
MICROPROGRAMMED CONTROL

- Control Memory
- Sequencing Microinstructions
- Microprogram Example
- Design of Control Unit
- Microinstruction Format
- Nanostorage and Nanoprogram

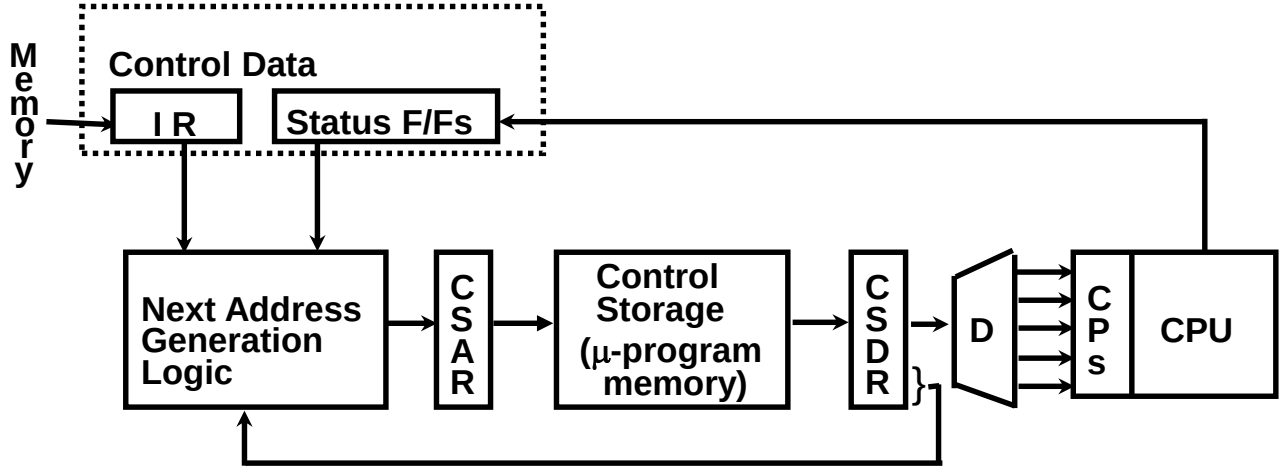
COMPARISON OF CONTROL UNIT IMPLEMENTATIONS

Control Unit Implementation

Combinational Logic Circuits (Hard-wired)



Microprogram



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

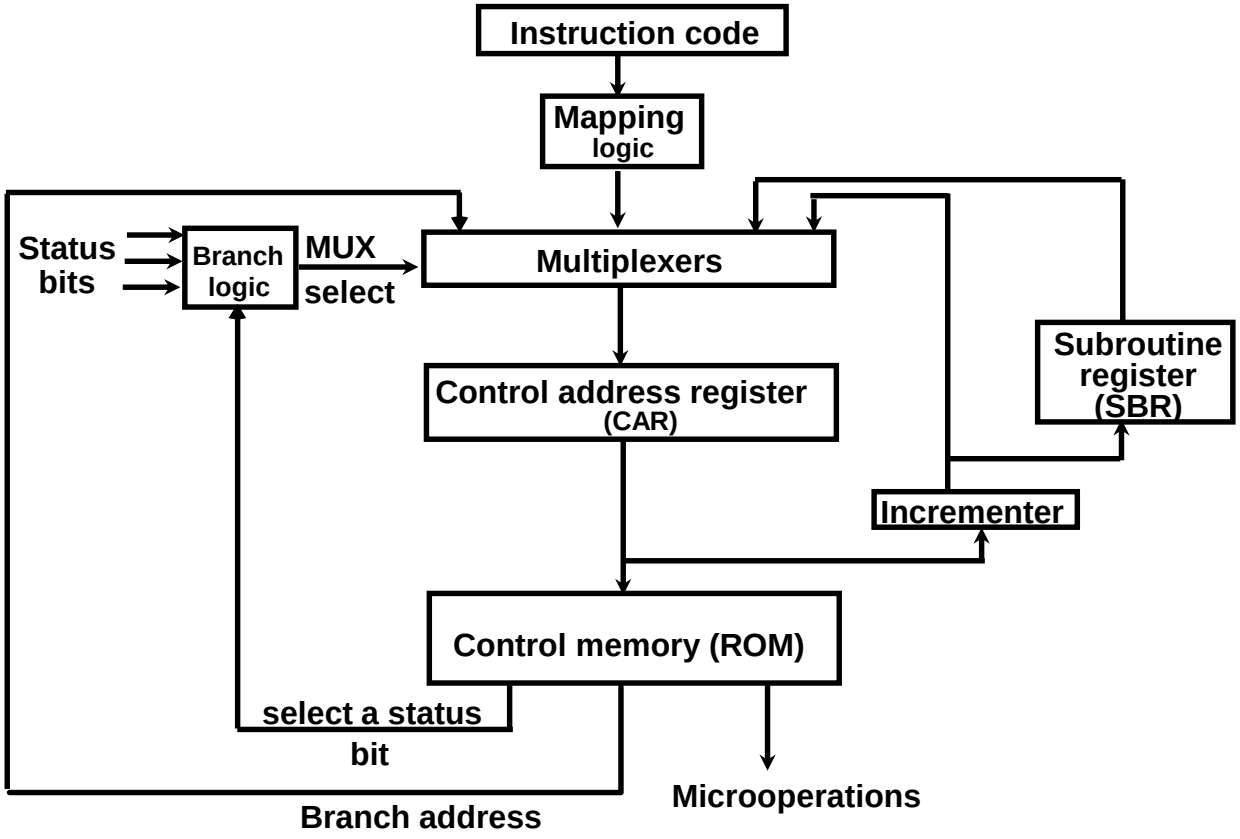
TERMINOLOGY

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

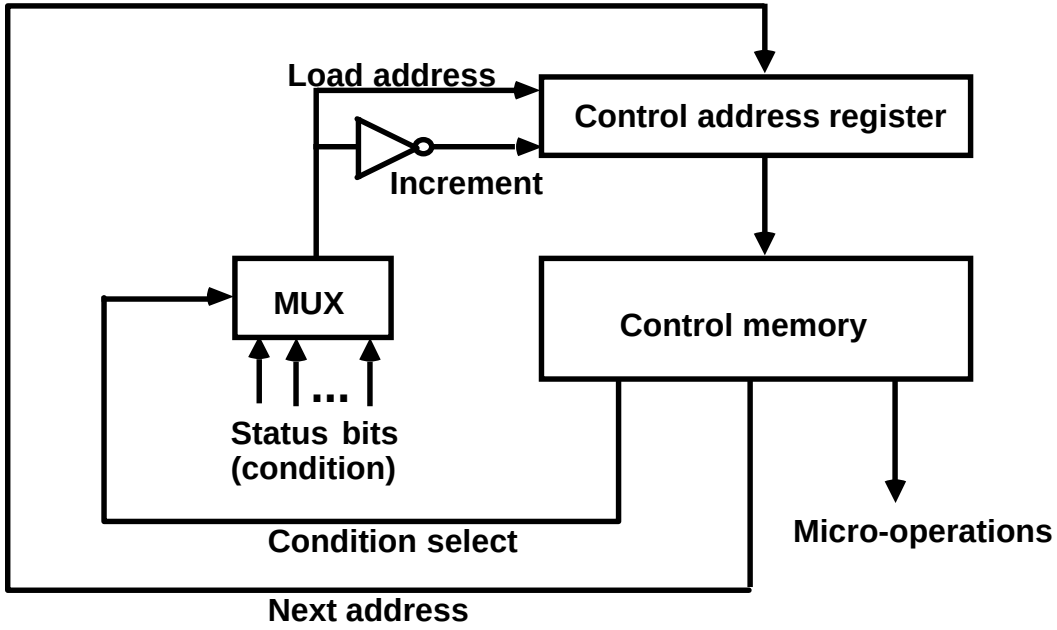
MICROINSTRUCTION SEQUENCING



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

CONDITIONAL BRANCH



Conditional Branch

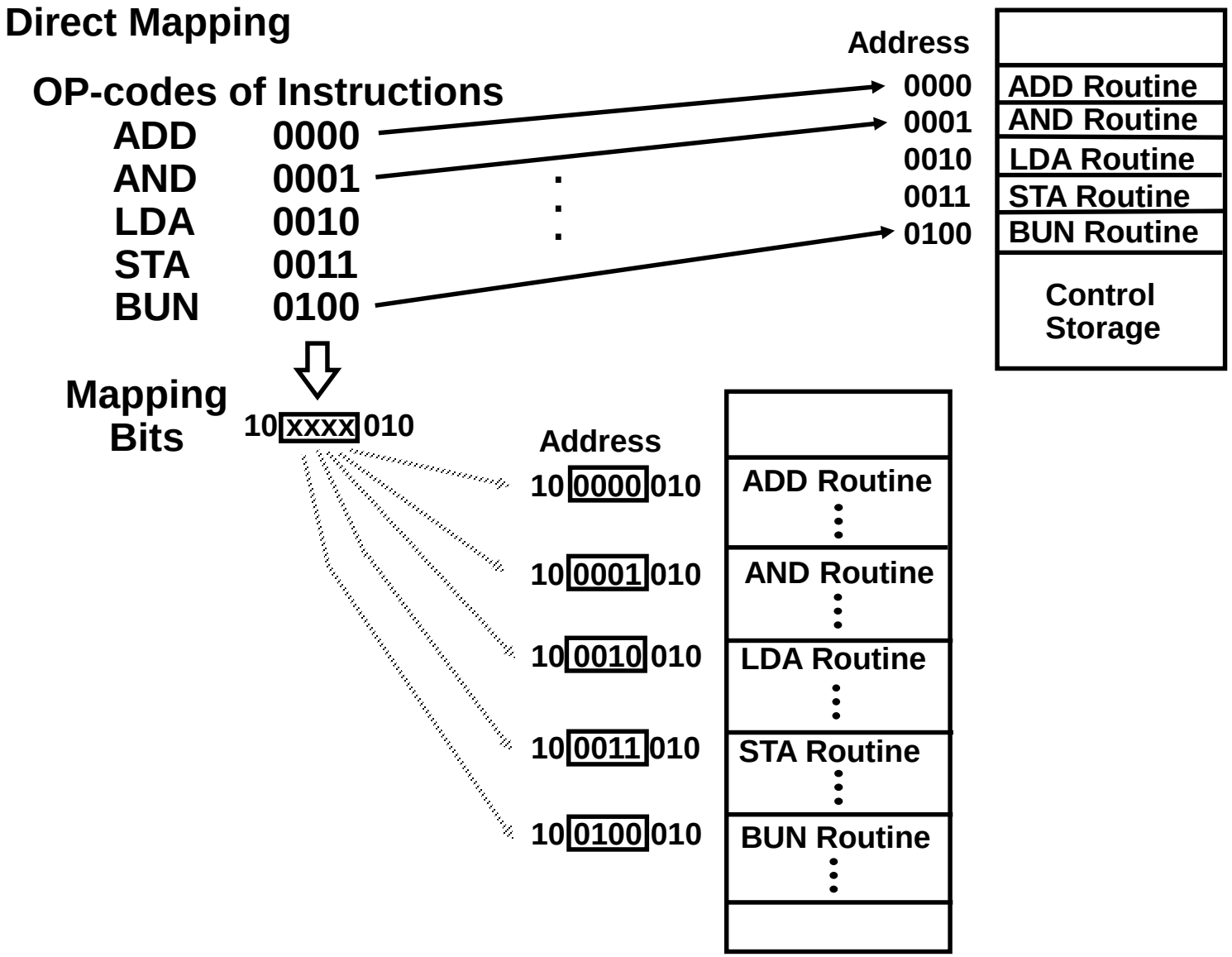
If *Condition* is true, then *Branch* (address from the next address field of the current microinstruction) else *Fall Through*

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

Unconditional Branch

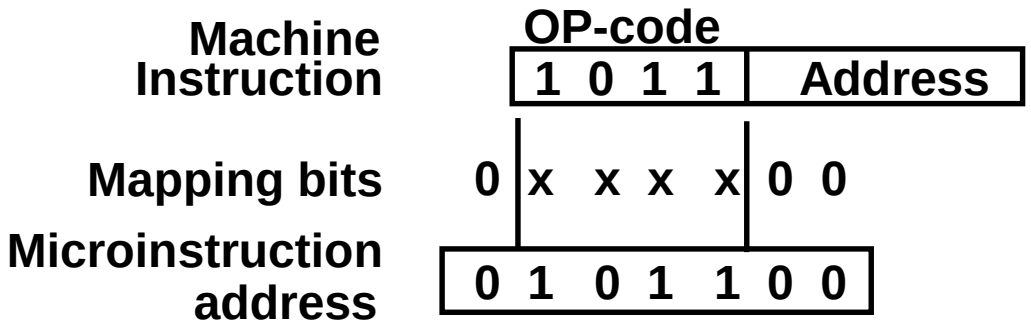
Fixing the value of one status bit at the input of the multiplexer to 1

MAPPING OF INSTRUCTIONS

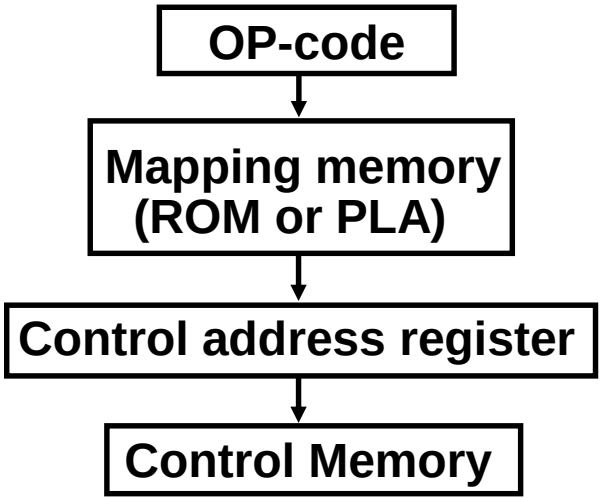


MAPPING OF INSTRUCTIONS TO MICROROUTINES

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

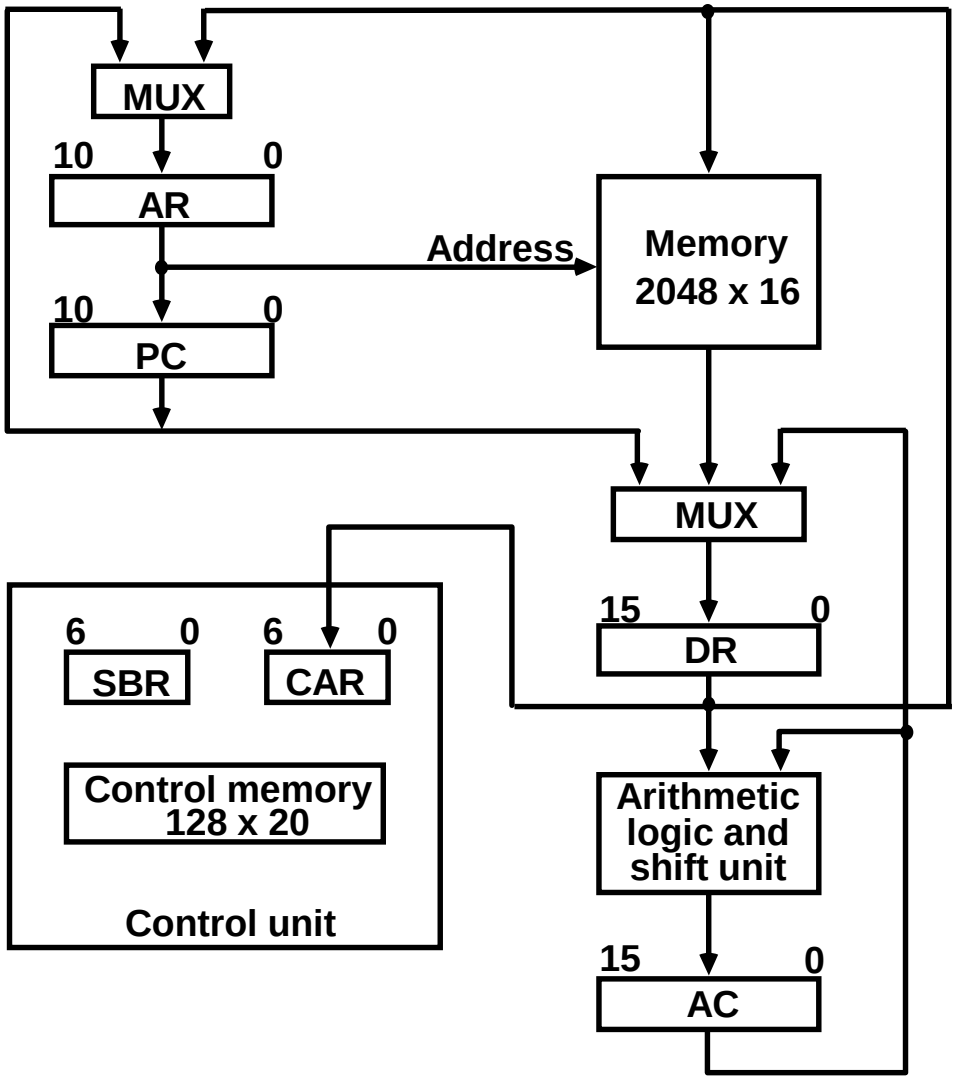


Mapping function implemented by ROM or PLA



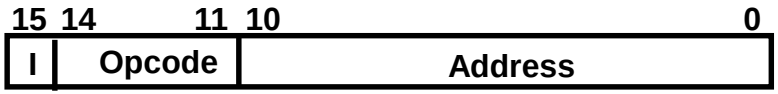
MICROPROGRAM EXAMPLE

Computer Configuration



MACHINE INSTRUCTION FORMAT

Machine instruction format

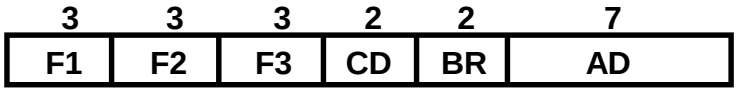


Sample machine instructions

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

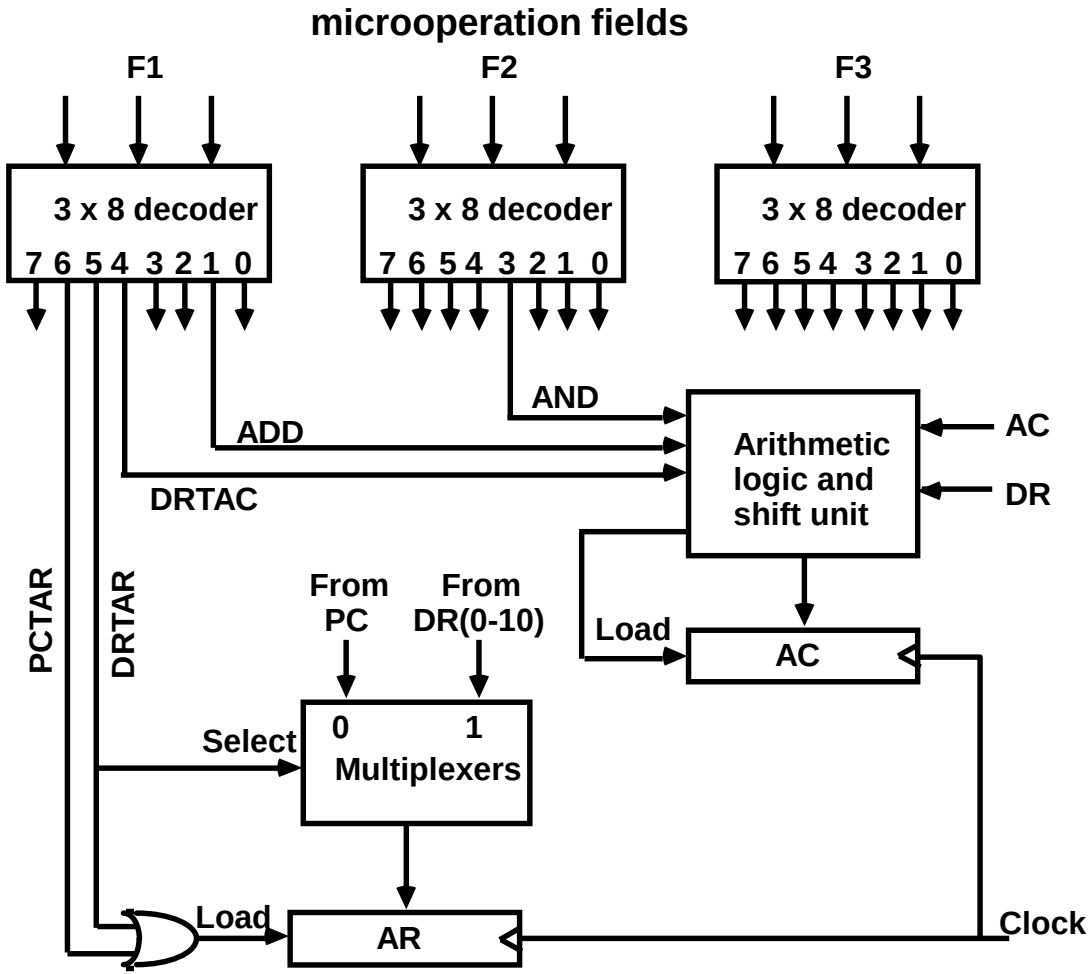
Label	Microops	CD	BR	AD
	ORG 0			
ADD:	NOP	I	CALL	INDRCT
	READ	U	JMP	NEXT
	ADD	U	JMP	FETCH
	ORG 4			
BRANCH:	NOP	S	JMP	OVER
	NOP	U	JMP	FETCH
OVER:	NOP	I	CALL	INDRCT
	ARTPC	U	JMP	FETCH
	ORG 8			
STORE:	NOP	I	CALL	INDRCT
	ACTDR	U	JMP	NEXT
	WRITE	U	JMP	FETCH
	ORG 12			
EXCHANGE:	NOP	I	CALL	INDRCT
	READ	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		Binary Microinstruction					
	Decimal	Binary	F1	F2	F3	CD	BR	AD
ADD	0	0000000	000	000	000	01	01	1000011
	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	111	000	000	00	00	1000000
FETCH	64	1000000	110	000	000	00	00	1000001
	65	1000001	000	100	101	00	00	1000010
	66	1000010	101	000	000	00	11	0000000
INDRCT	67	1000011	000	100	000	00	00	1000100
	68	1000100	101	000	000	00	10	0000000

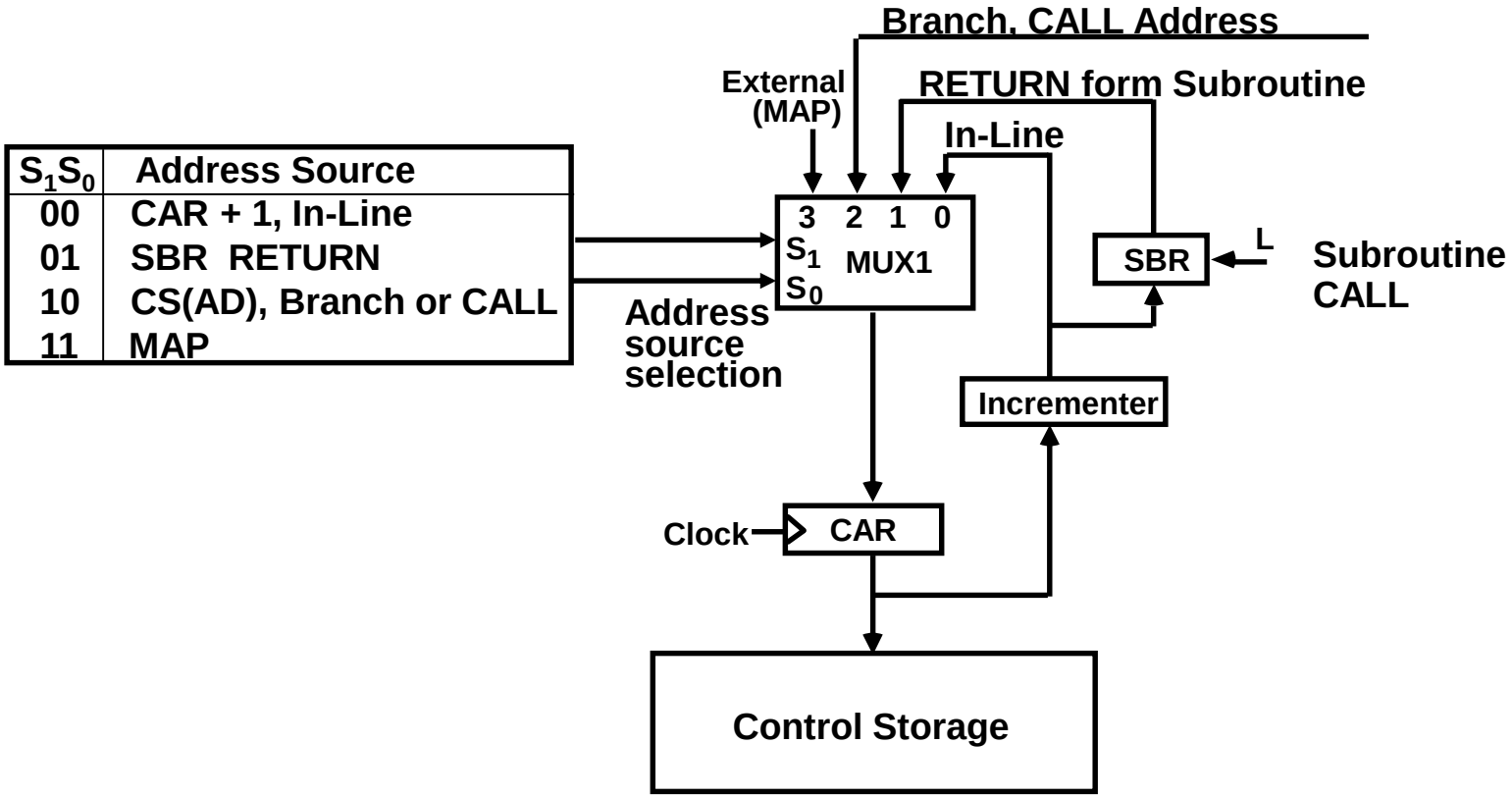
This microprogram can be implemented using ROM

DESIGN OF CONTROL UNIT
- DECODING ALU CONTROL INFORMATION -



Decoding of Microoperation Fields

MICROPROGRAM SEQUENCER
- NEXT MICROINSTRUCTION ADDRESS LOGIC -

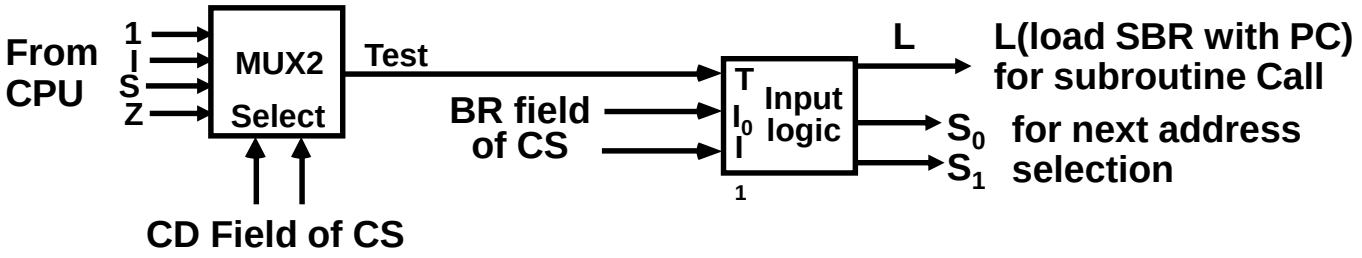


S_1S_0	Address Source
00	CAR + 1, In-Line
01	SBR RETURN
10	CS(AD), Branch or CALL
11	MAP

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 -

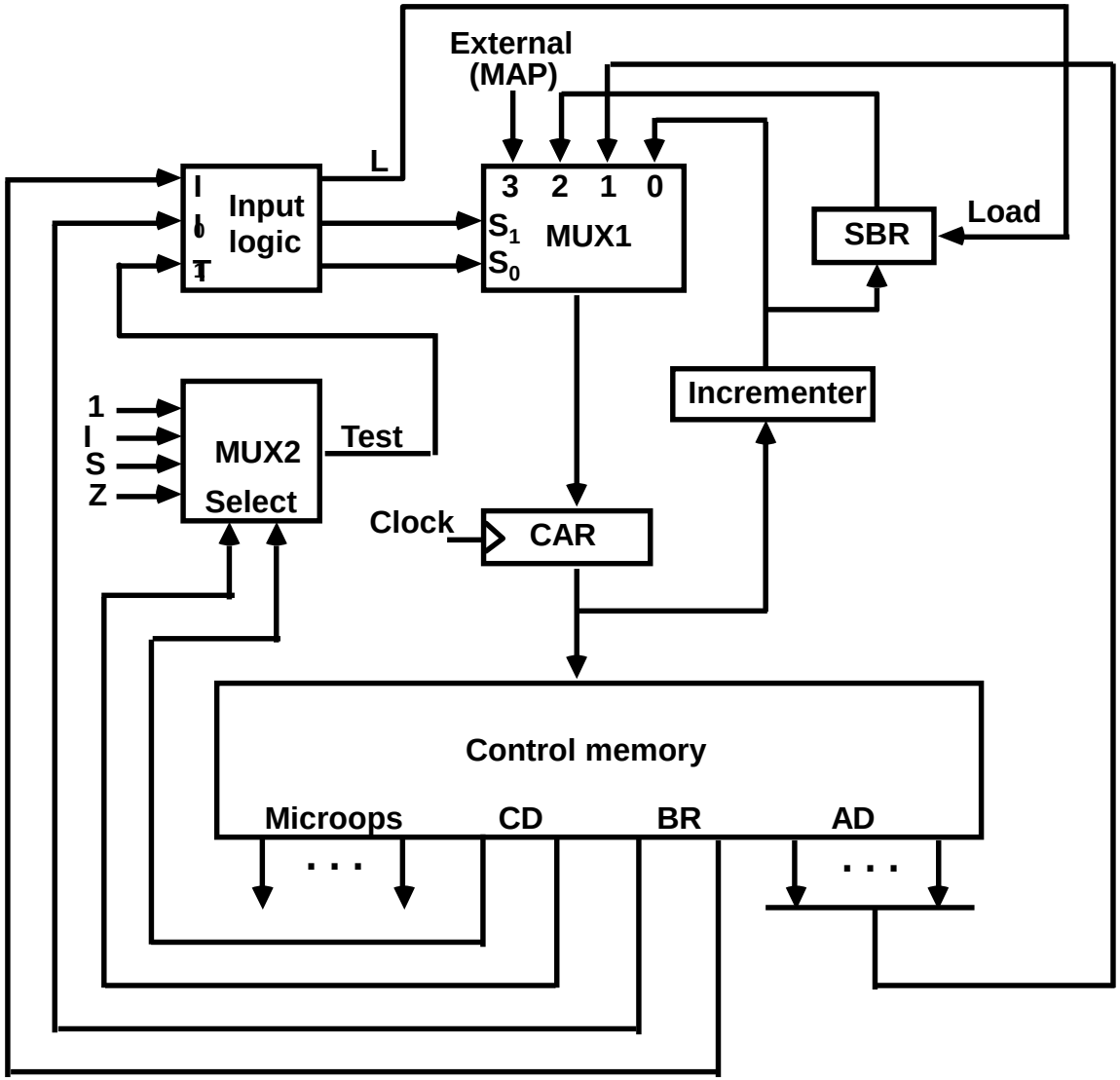


Input Logic

I_1I_0T	Meaning	Source of Address	S_1S_0	L
000	In-Line	CAR+1	00	0
001	JMP	CS(AD)	01	0
010	In-Line	CAR+1	00	0
011	CALL	CS(AD) and SBR <- CAR+1	01	1
10x	RET	SBR	10	0
11x	MAP	DR(11-14)	11	0

$$S_1 = I_1$$
$$S_0 = I_1I_0 + I_1'T$$
$$L = I_1'I_0T$$

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

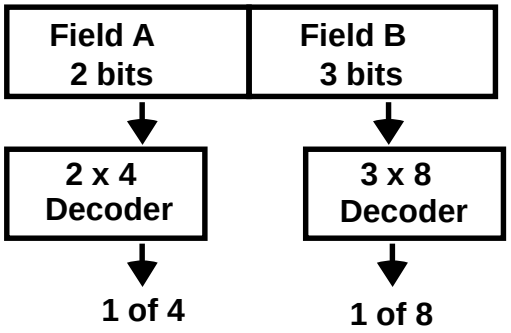
Horizontal Microinstructions

- 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

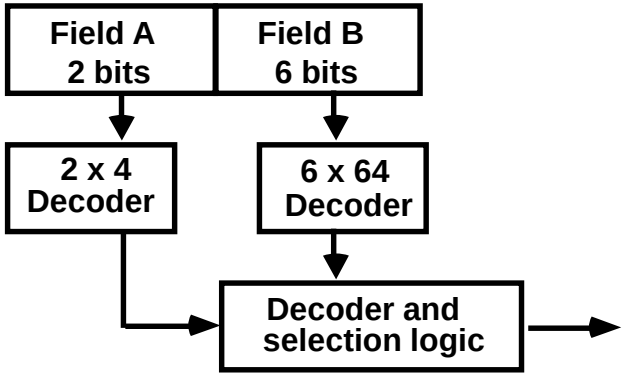
Vertical Microinstructions

- 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)