BLG222E
Computer Organization
Recitation 2

Spring 2024

23.05.2024

QUESTION 1-a

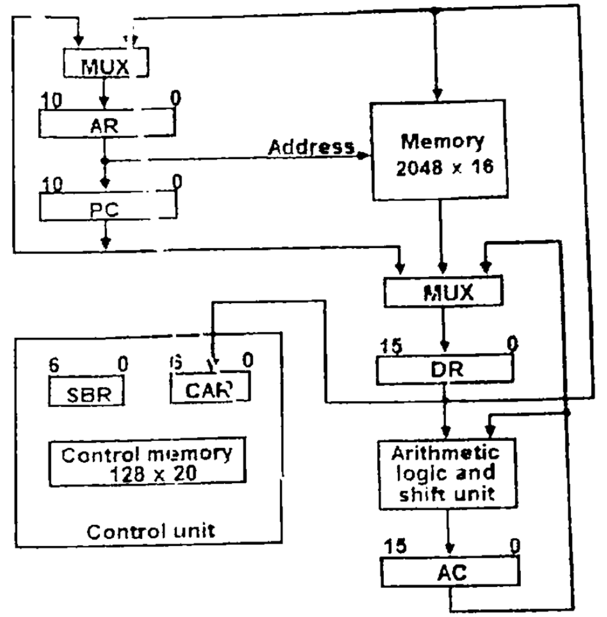
(a) In accord with the given CPU organization, implement memory subtract (MES) that subtracts the value of AC register from the given address at the memory, and write the result to the next address:

M[EA + 1] ← M[EA] - AC

Assume that the opcode for MES is 0100.

Ignore fetch and decode cycles and assume that the effective address is already in AR register (i.e. ignore indirect addressing mode).

Write a micro-program that will implement MES instruction. Use the microinstruction format that is given in Figure 1, and mapping algorithm given in Figure 2.



3	3	3	2	2	7
F1	F2	F3	CD	BR	AD

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

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	AR ← AR + 1	INCAR

Figure 1: Microinstruction format and microoperations.

0	4-bit opcode	0	0

Figure 2: Mapping algorithm.

SOLUTION 1-a

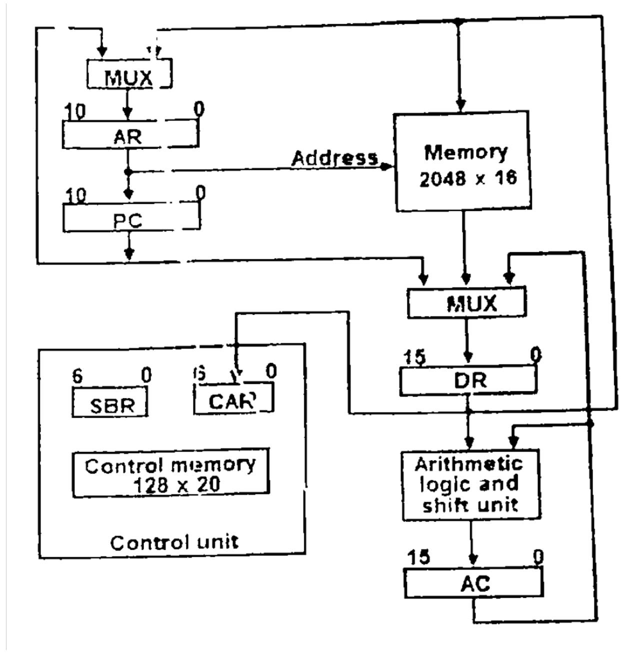
$$M[EA+1] \leftarrow M[EA] - AC$$

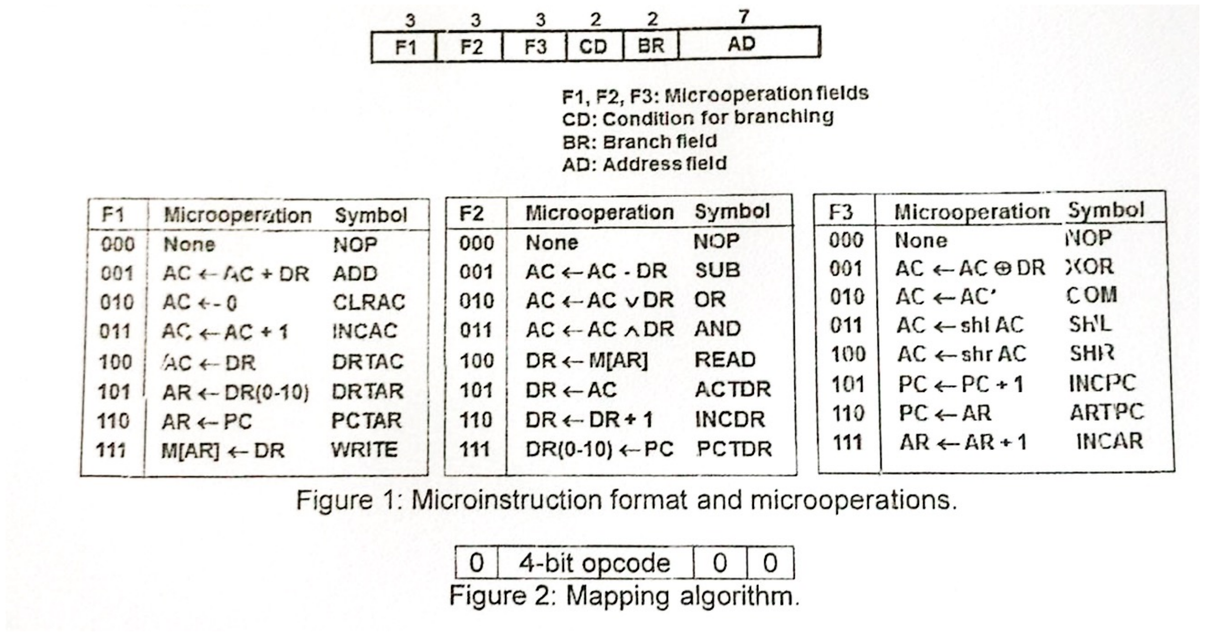
$$\left.\begin{array}{l} DR \leftarrow M[AR], AC \leftarrow AC' \\ AC \leftarrow AC+1, AR \leftarrow AR+1 \end{array}\right\}$$ 2's complement of AC

$$AC \leftarrow AC + DR$$

$$DR \leftarrow AC$$

$$M[AR] \leftarrow DR$$





3	3	3	2	2	7
F1	F2	F3	CD	BR	AD

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

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	AR ← AR + 1	INCAR

Figure 1: Microinstruction format and microoperations.

0	4-bit opcode	0	0

Figure 2: Mapping algorithm.

SOLUTION 1-a

0010000 = 16 (decimal)

$DR \leftarrow M[AR], AC \leftarrow AC'$

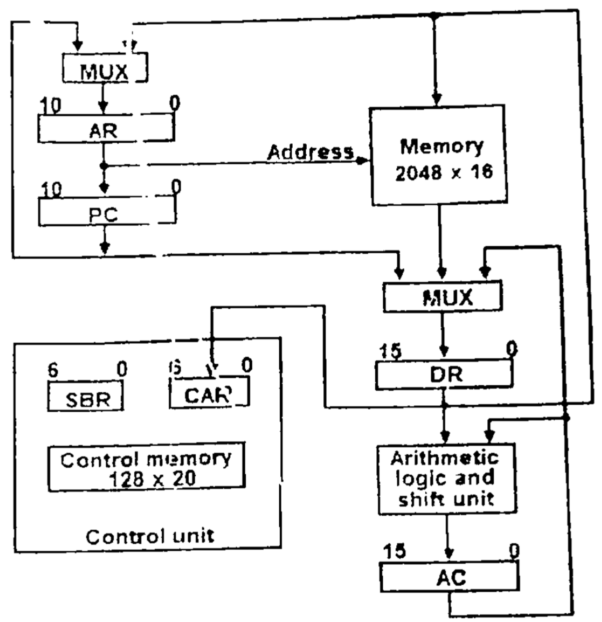
$AC \leftarrow AC + 1, AR \leftarrow AR + 1$

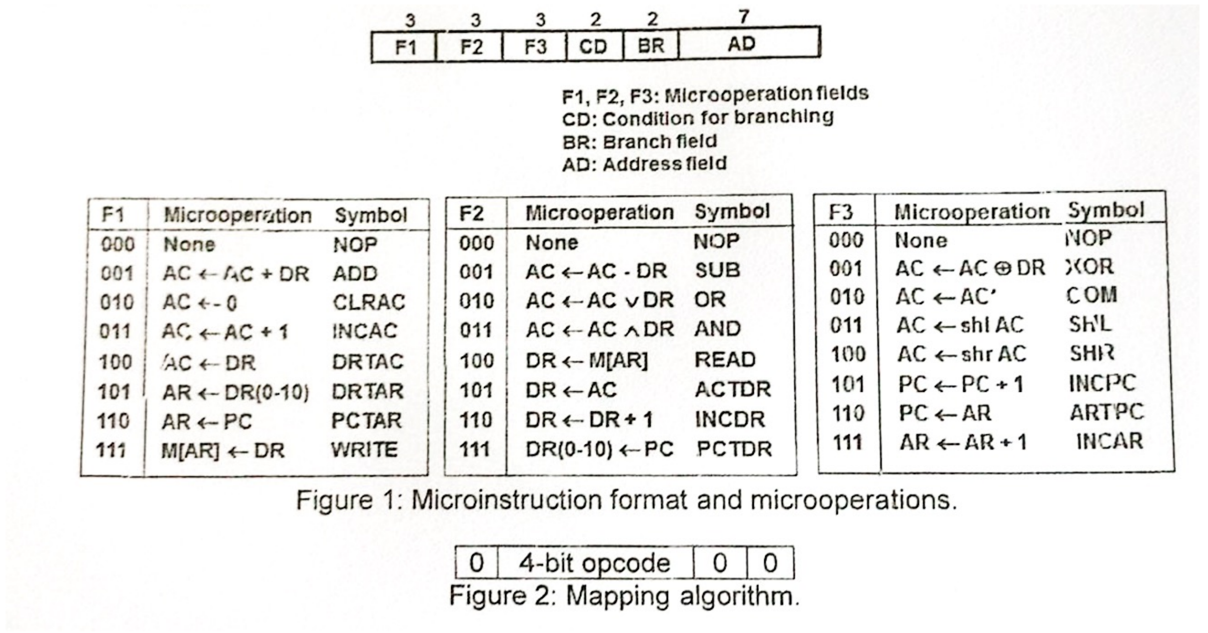
$AC \leftarrow AC + DR$

$DR \leftarrow AC$

$M[AR] \leftarrow DR$

```
        ORG  16
MES:    READ, COM       U   JMP   NEXT
        INCAC, INCAR    U   JMP   NEXT
        ADD             U   JMP   NEXT
        ACTDR           U   JMP   NEXT
        WRITE           U   JMP   FETCH
```





3	3	3	2	2	7
F1	F2	F3	CD	BR	AD

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

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	AR ← AR + 1	INCAR

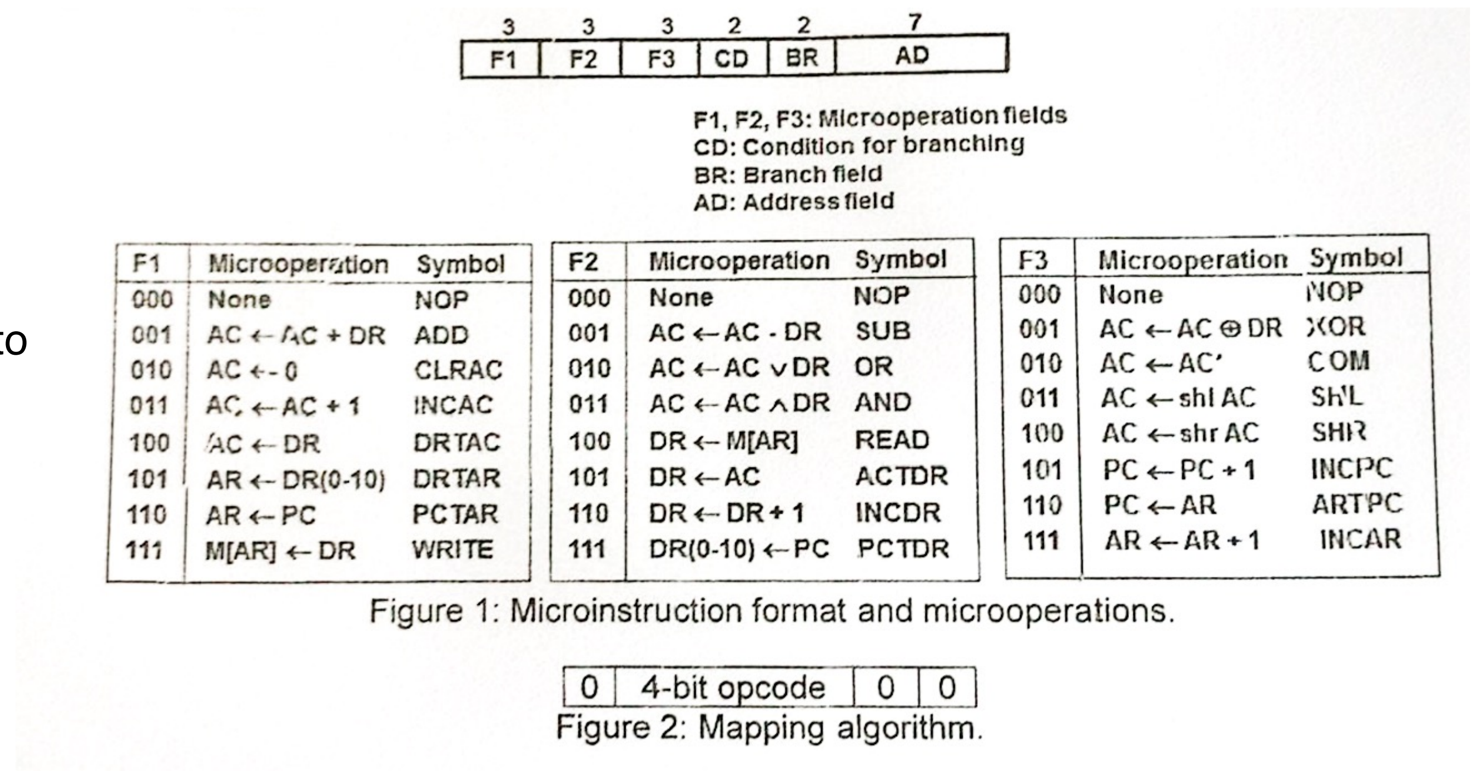
Figure 1: Microinstruction format and microoperations.

0	4-bit opcode	0	0

Figure 2: Mapping algorithm.

QUESTION 1-b

(b) Convert your micro-program from part (a) into machine codes using codes that are given in Figures 1 and 3.



3	3	3	2	2	7
F1	F2	F3	CD	BR	AD

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

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

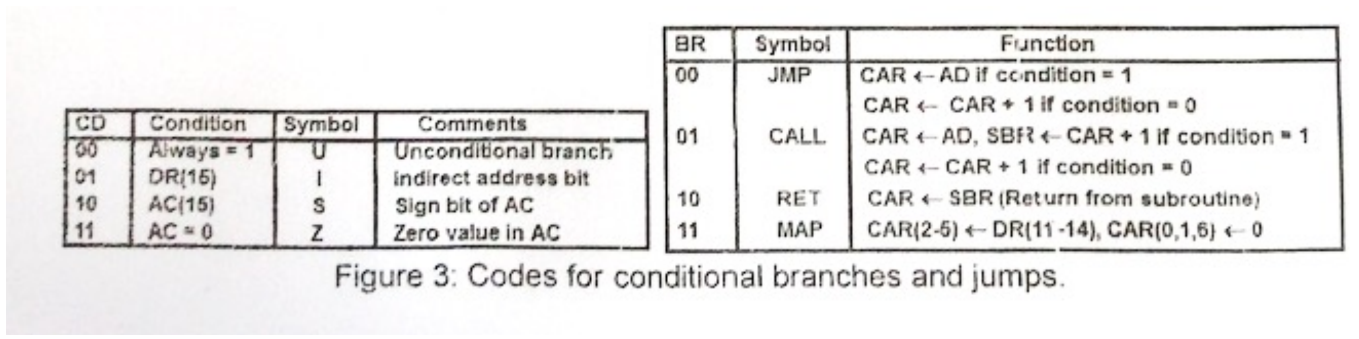
F2	Microoperation	Symbol
000	None	NOP
001	AC ← AC - DR	SUB
010	AC ← AC ∨ DR	OR
011	AC ← AC ∧ DR	AND
100	DR ← M[AR]	READ
101	DR ← AC	ACTDR
110	DR ← DR + 1	INCDR
111	DR(0-10) ← PC	PCTDR

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	AR ← AR + 1	INCAR

Figure 1: Microinstruction format and microoperations.

0	4-bit opcode	0	0

Figure 2: Mapping algorithm.



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

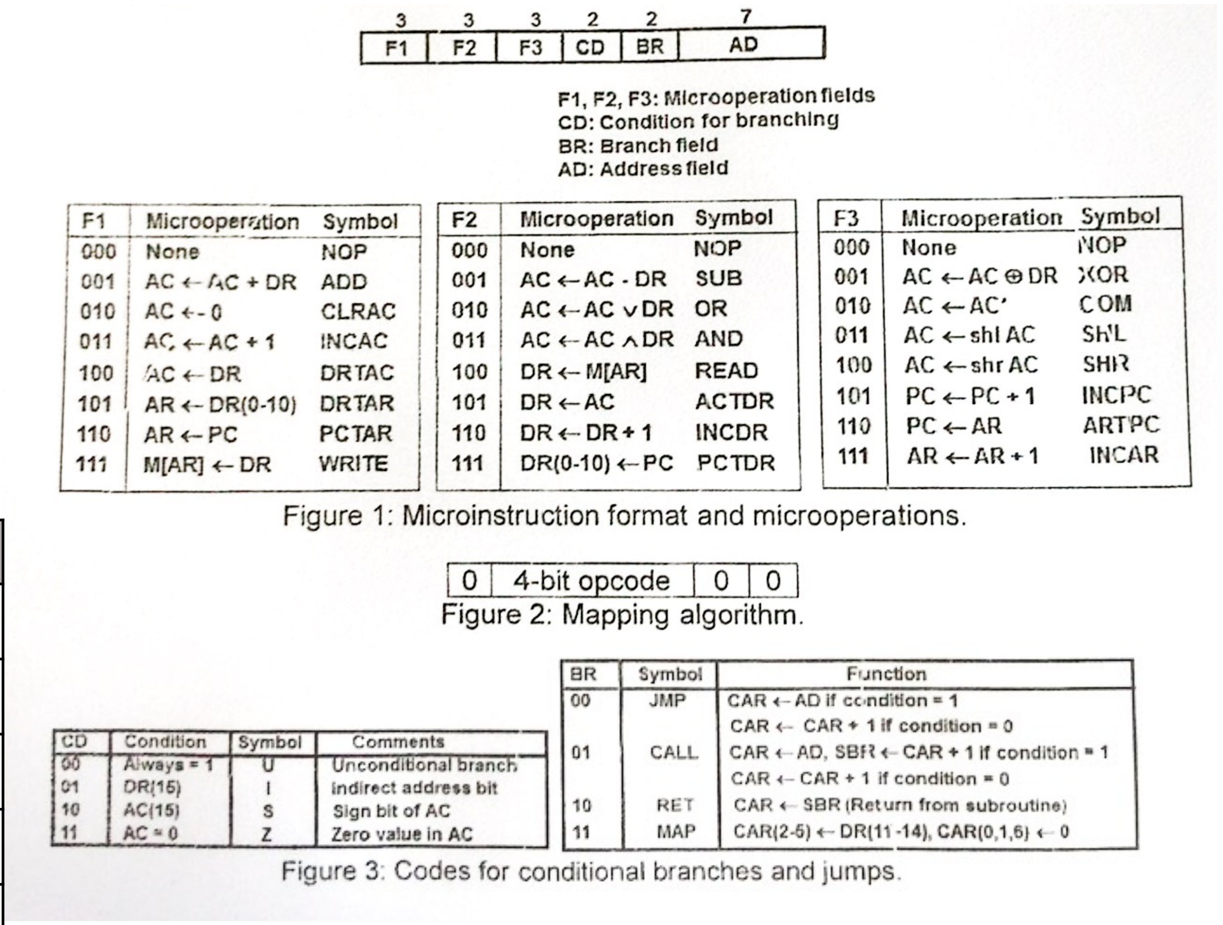
01	CALL	CAR ← AD, SBR ← CAR + 1 if condition = 1
		CAR ← CAR + 1 if condition = 0

10	RET	CAR ← SBR (Return from subroutine)
11	MAP	CAR(2-5) ← DR(11-14), CAR(0,1,6) ← 0

Figure 3: Codes for conditional branches and jumps.

SOLUTION 1-b

Index	Address (7)	F1 (3)	F2 (3)	F3 (3)	CD (2)	BR (2)	AD (7)
1	0010000	000	100	010	00	00	0010001
2	0010001	011	000	111	00	00	0010010
3	0010010	001	000	000	00	00	0010011
4	0010011	000	101	000	00	00	0010100
5	0010100	111	000	000	00	00	1000000



3	3	3	2	2	7
F1	F2	F3	CD	BR	AD

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

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	AR ← AR + 1	INCAR

Figure 1: Microinstruction format and microoperations.

0	4-bit opcode	0	0

Figure 2: Mapping algorithm.

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

Figure 3: Codes for conditional branches and jumps.

$$DR \leftarrow M[AR], AC \leftarrow AC'$$

$$AC \leftarrow AC + 1, AR \leftarrow AR + 1$$

$$AC \leftarrow AC + DR$$

$$DR \leftarrow AC$$

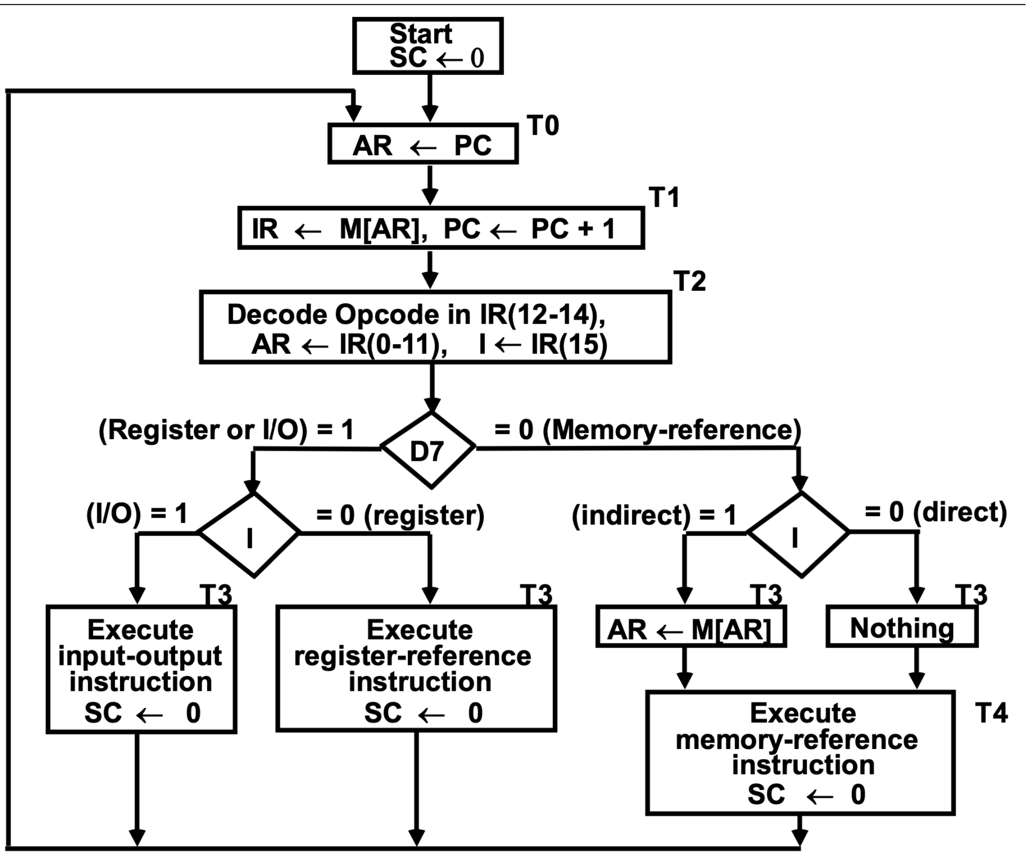
$$M[AR] \leftarrow DR$$

QUESTION 2

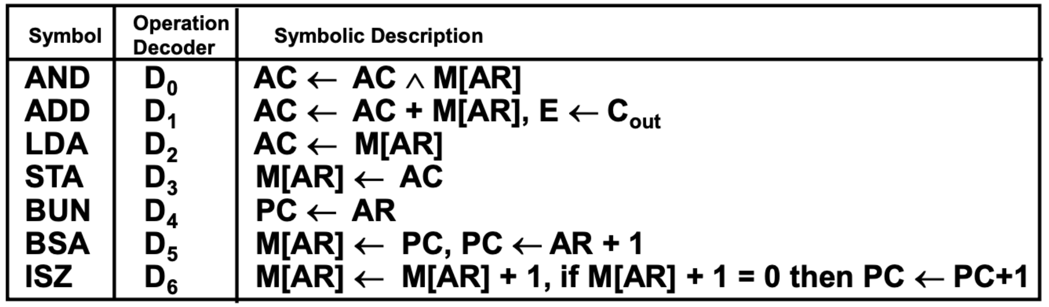
Show the content of registers PC, AR, IR, and SC of the basic computer in hexadecimal when the
following instruction is fetched from memory and executed.

- The initial content of PC is 1B1.
- The content of memory at address 1B1 is 52FF.
- The content of memory at address 2FF is 0B43.
- The content of memory at address B43 is 0DFF.

Show the contents of the registers along with the corresponding RTL statements after the positive
transition of each clock pulse using a chart.



Start
SC ← 0
T0
AR ← PC
T1
IR ← M[AR], PC ← PC + 1
T2
Decode Opcode in IR(12-14),
AR ← IR(0-11), I ← IR(15)
(Register or I/O) = 1
D7
= 0 (Memory-reference)
(I/O) = 1
I
= 0 (register)
(indirect) = 1
I
= 0 (direct)
T3
T3
T3
T3
Execute input-output instruction SC ← 0
Execute register-reference instruction SC ← 0
AR ← M[AR]
Nothing
T4
Execute memory-reference instruction SC ← 0



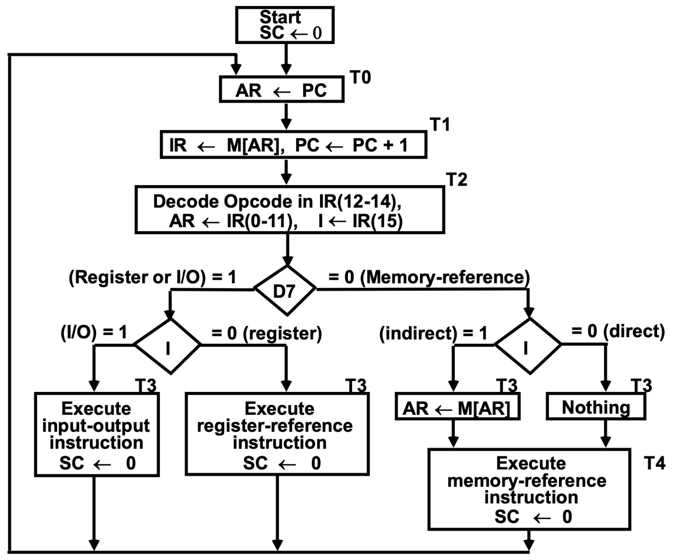
Symbol	Operation Decoder	Symbolic Description
AND	D_0	$AC \leftarrow AC \wedge M[AR]$
ADD	D_1	$AC \leftarrow AC + M[AR], E \leftarrow C_{out}$
LDA	D_2	$AC \leftarrow M[AR]$
STA	D_3	$M[AR] \leftarrow AC$
BUN	D_4	$PC \leftarrow AR$
BSA	D_5	$M[AR] \leftarrow PC, PC \leftarrow AR + 1$
ISZ	D_6	$M[AR] \leftarrow M[AR] + 1$, if $M[AR] + 1 = 0$ then $PC \leftarrow PC+1$

SOLUTION 2

Time	Microoperation	PC	AR	IR	I	SC
T0	AR ← PC	1B1	1B1	-	-	1
T1	IR ← M[AR], PC←PC+1	1B2	1B1	52FF	-	2
T2	Decode ← IR[12-14] AR ← IR[0-11] I ← IR[15]	1B2	2FF	52FF	0	3
T3	Nothing	1B2	2FF	52FF	0	4
T4	M[AR] ← PC AR ← AR +1	1B2	300	52FF	0	5
T4	PC ← AR, SC ← 0	300	300	52FF	0	0

MEMORY	
Address	Content
1B1	52FF
…	…
2FF	0B43
…	…
B43	0DFF

PC	1B1

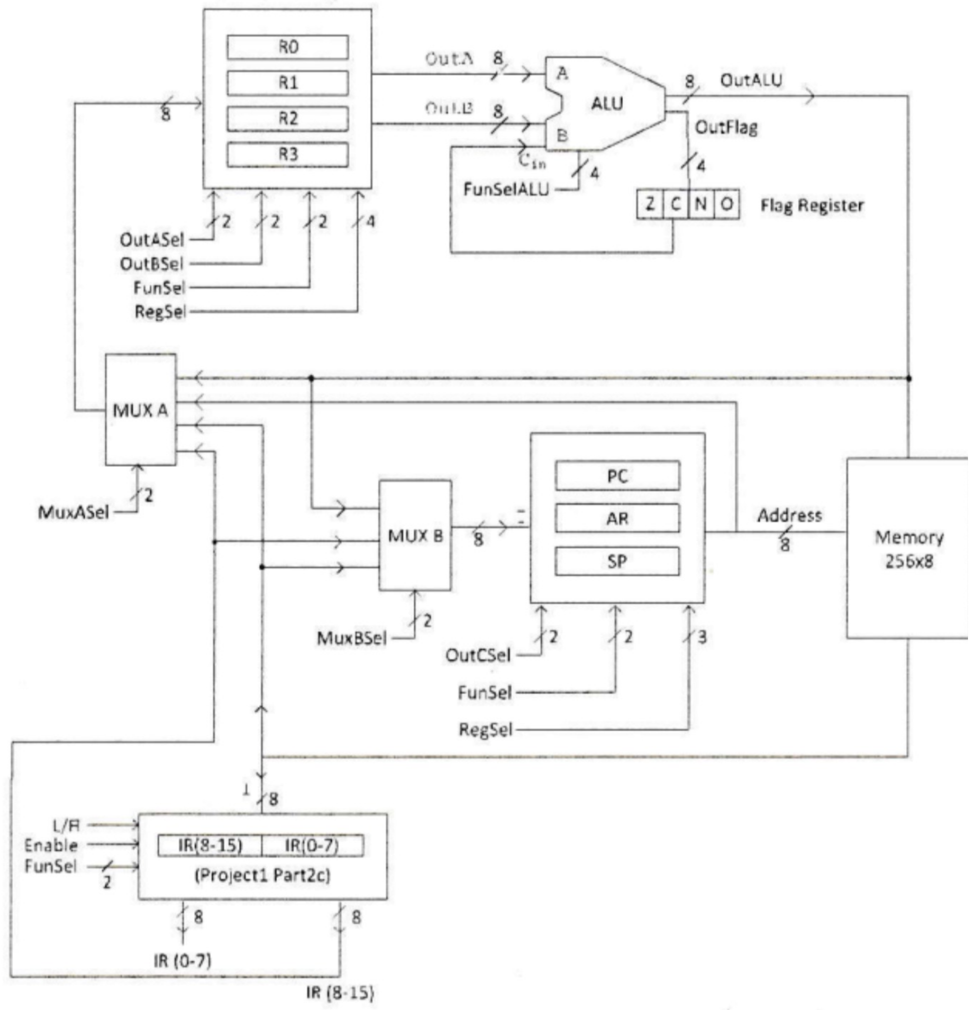


Start
SC ← 0
AR ← PC
T0
IR ← M[AR], PC ← PC + 1
T1
Decode Opcode in IR(12-14),
AR ← IR(0-11), I ← IR(15)
T2
(Register or I/O) = 1
D7
= 0 (Memory-reference)
(I/O) = 1
I
= 0 (register)
(indirect) = 1
I
= 0 (direct)
T3
Execute input-output instruction SC ← 0
T3
Execute register-reference instruction SC ← 0
T3
AR ← M[AR]
T3
Nothing
T4
Execute memory-reference instruction SC ← 0

BSA = Branch and Save Address

Symbol	Operation Decoder	Symbolic Description
AND	D_0	AC ← AC ∧ M[AR]
ADD	D_1	AC ← AC + M[AR], E ← C_{out}
LDA	D_2	AC ← M[AR]
STA	D_3	M[AR] ← AC
BUN	D_4	PC ← AR
BSA	D_5	M[AR] ← PC, PC ← AR + 1
ISZ	D_6	M[AR] ← M[AR] + 1, if M[AR] + 1 = 0 then PC ← PC+1

QUESTION 3-a



Consider the instruction 0x1D00. Describe very shortly, what this instruction performs.

Control signals for register files:

FunSel	R_x^+
00	0
01	$R_x + 1$
10	$R_x - 1$
11	I (load)

OutA(B)Sel	OutA(B)	OutCSel	OutC
00	R0	00	PC
01	R1	01	PC
10	R2	10	AR
11	R3	11	SP

Control signals for IR register:

$L/\bar{H}$	Enable	FunSel	IR^+
Φ	0	Φ	IR
1	1	11	IR(0-7)←I
0	1	11	IR(8-15)←I
Φ	1	00	0
Φ	1	01	IR+1
Φ	1	10	IR-1

Selection signals for MUXA and MUXB:

MUXASel	MuxAOut	MUXBSel	MuxBOut
00	OutALU	00	OutALU
01	Address	01	Φ
10	Memory	10	Memory
11	IR(8-15)	11	IR(8-15)

Instruction format:
For instructions with address reference
IR(15-11): opcode
IR(10-9): regsel
IR(8): Addressing Mode
IR(7-0): Address

For instructions without address reference
IR(15-11): opcode
IR(10-8): destreg
IR(7-5): srcreg1
IR(4-2): srcreg2
IR(1-0): Not used

Instruction set:

OPCODE	SYMB	ADDR MODE	DESCRIPTION
0x00	LD	IM, D	Rx ← Value (Value is described in Table 3)
0x01	ST	D	Value ← Rx
0x02	MOV	N/A	DESTREG ← SRCREG1
0x03	PSH	N/A	M[SP] ← Rx, SP ← SP - 1
0x04	PUL	N/A	SP ← SP + 1, Rx ← M[SP]
0x05	ADD	N/A	DESTREG ← SRCREG1 + SRCREG2
0x06	SUB	N/A	DESTREG ← SRCREG2 - SRCREG1
0x07	DEC	N/A	DESTREG ← SRCREG1 - 1
0x08	INC	N/A	DESTREG ← SRCREG1 + 1
0x09	AND	N/A	DESTREG ← SRCREG1 AND SRCREG2
0x0A	OR	N/A	DESTREG ← SRCREG1 OR SRCREG2
0x0B	NOT	N/A	DESTREG ← NOT SRCREG1
0x0C	LSL	N/A	DESTREG ← LSL SRCREG1
0x0D	LSR	N/A	DESTREG ← LSR SRCREG1
0x0E	BRA	IM	PC ← Value
0x0F	BEQ	IM	IF Z=1 THEN PC ← Value
0x10	BNE	IM	IF Z=0 THEN PC ← Value
0x11	CALL	IM	M[SP] ← PC, SP ← SP – 1, PC ← Value
0x12	RET	N/A	SP ← SP + 1, PC ← M[SP]

Description of fields in the instruction set:

REGSEL	REGISTER
00	R0
01	R1
10	R2
11	R3

DESTREG/SRCREG1/SRCREG2	REGISTER
000	R0
001	R1
010	R2
011	R3
100	PC
101	PC
110	AR
111	SP

Addressing modes:

ADDRESSING MODE	MODE	SYMB	Value
0	Immediate	IM	ADDRESS Field
1	Direct	D	M[AR]

SOLUTION 3-a

0x1D00 = 0001 1101 0000 0000

Opcode: 0x03 = PSH (no address mode)

Destreg: 101

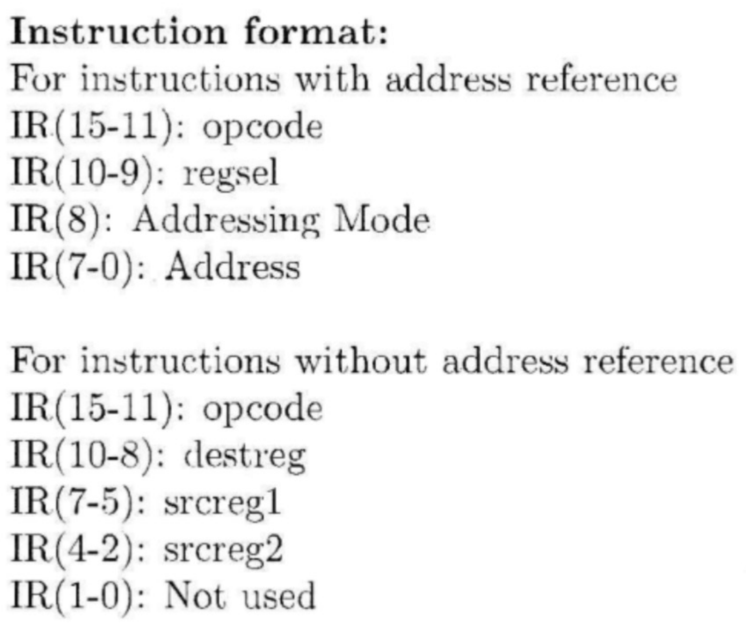
Srcreg1: 000 → R0

Srcreg2: 000

Not used: 00

M[SP] ← R0, SP ← SP - 1

Push content of R0 to Stack.



Instruction format:

For instructions with address reference

IR(15-11): opcode

IR(10-9): regsel

IR(8): Addressing Mode

IR(7-0): Address

For instructions without address reference

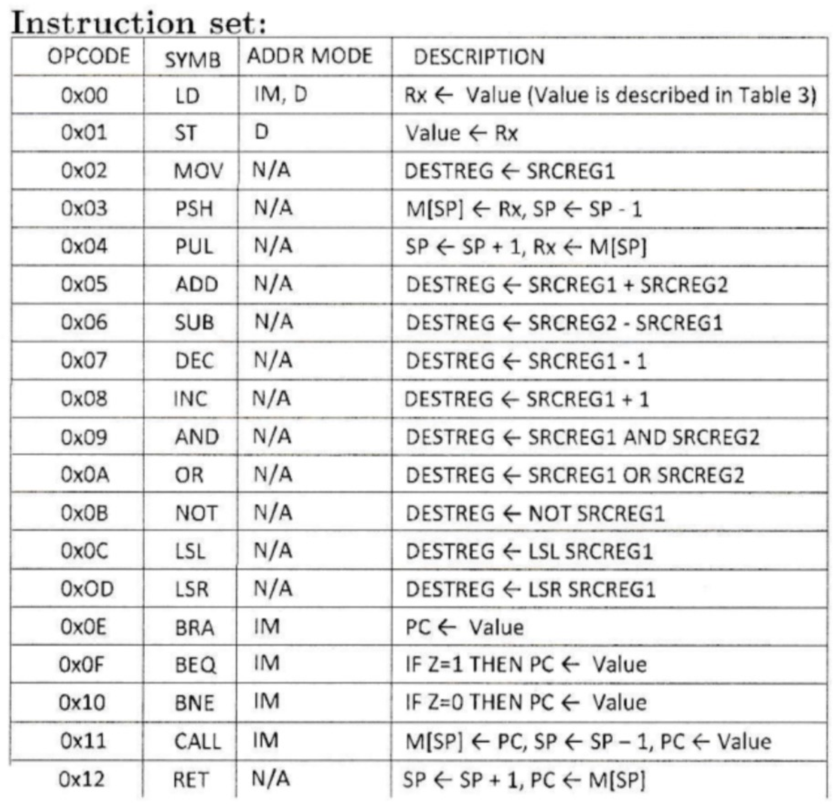
IR(15-11): opcode

IR(10-8): destreg

IR(7-5): srcreg1

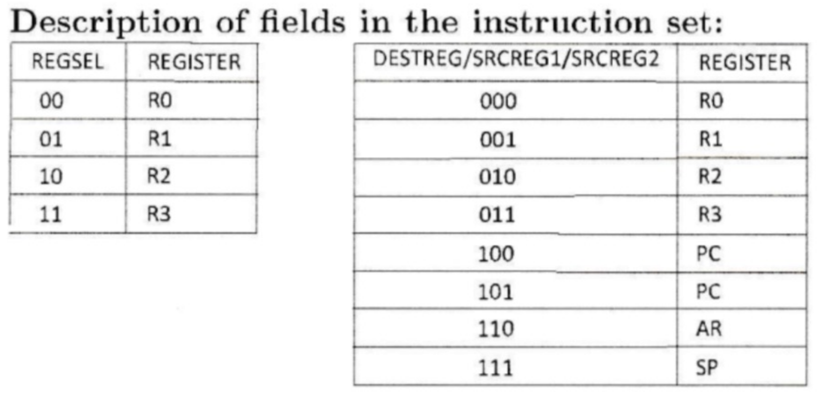
IR(4-2): srcreg2

IR(1-0): Not used



Instruction set:

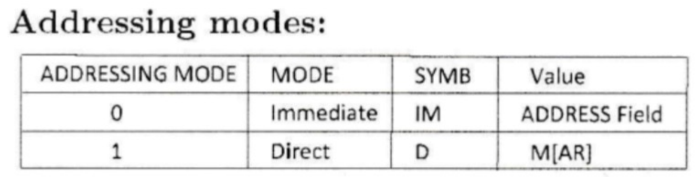
OPCODE	SYMB	ADDR MODE	DESCRIPTION
0x00	LD	IM, D	Rx ← Value (Value is described in Table 3)
0x01	ST	D	Value ← Rx
0x02	MOV	N/A	DESTREG ← SRCREG1
0x03	PSH	N/A	M[SP] ← Rx, SP ← SP - 1
0x04	PUL	N/A	SP ← SP + 1, Rx ← M[SP]
0x05	ADD	N/A	DESTREG ← SRCREG1 + SRCREG2
0x06	SUB	N/A	DESTREG ← SRCREG2 - SRCREG1
0x07	DEC	N/A	DESTREG ← SRCREG1 - 1
0x08	INC	N/A	DESTREG ← SRCREG1 + 1
0x09	AND	N/A	DESTREG ← SRCREG1 AND SRCREG2
0x0A	OR	N/A	DESTREG ← SRCREG1 OR SRCREG2
0x0B	NOT	N/A	DESTREG ← NOT SRCREG1
0x0C	LSL	N/A	DESTREG ← LSL SRCREG1
0x0D	LSR	N/A	DESTREG ← LSR SRCREG1
0x0E	BRA	IM	PC ← Value
0x0F	BEQ	IM	IF Z=1 THEN PC ← Value
0x10	BNE	IM	IF Z=0 THEN PC ← Value
0x11	CALL	IM	M[SP] ← PC, SP ← SP – 1, PC ← Value
0x12	RET	N/A	SP ← SP + 1, PC ← M[SP]



Description of fields in the instruction set:

REGSEL	REGISTER
00	R0
01	R1
10	R2
11	R3

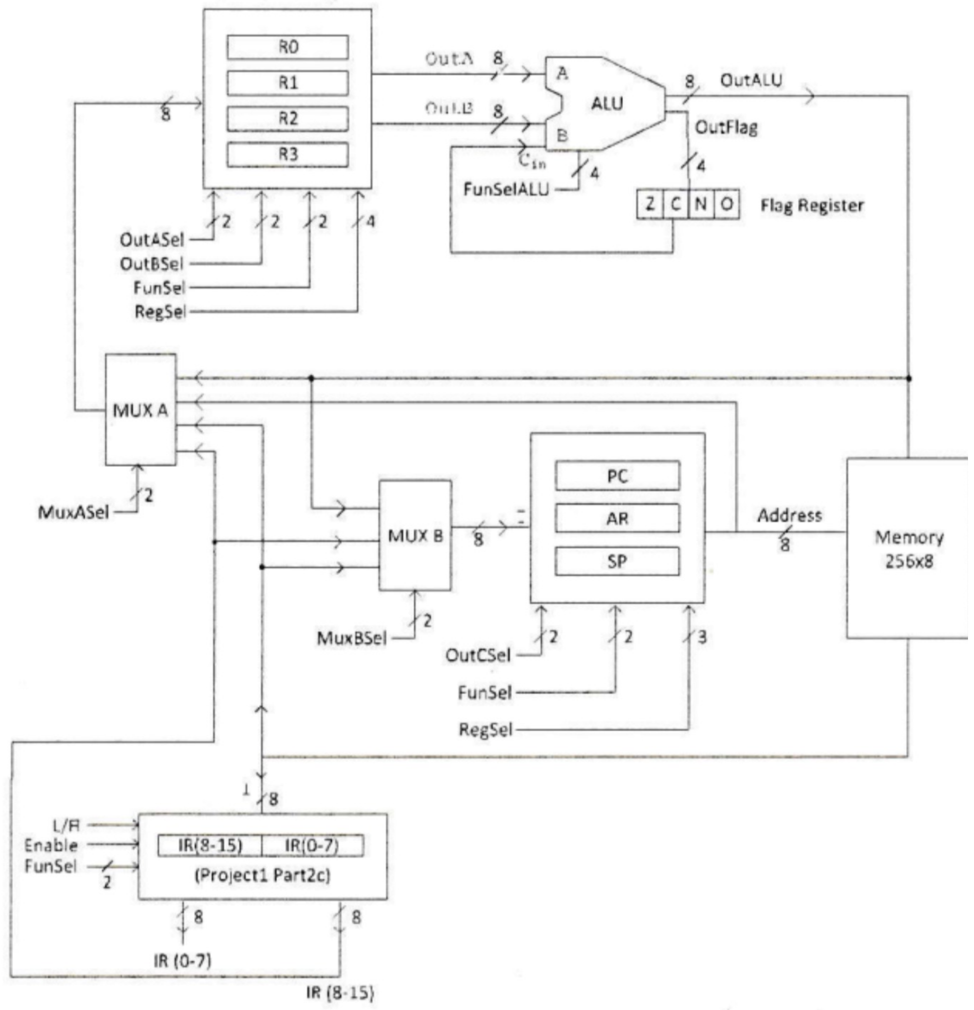
DESTREG/SRCREG1/SRCREG2	REGISTER
000	R0
001	R1
010	R2
011	R3
100	PC
101	PC
110	AR
111	SP



Addressing modes:

ADDRESSING MODE	MODE	SYMB	Value
0	Immediate	IM	ADDRESS Field
1	Direct	D	M[AR]

QUESTION 3-b



Write the sequence of microoperations using RTL that are required to exececute the instruction given on part a. Specify the values of relevant control signals.

Control signals for register files:

FunSel	R_x^+
00	0
01	$R_x + 1$
10	$R_x - 1$
11	I (load)

OutA(B)Sel	OutA(B)
00	R0
01	R1
10	R2
11	R3

OutCSel	OutC
00	PC
01	PC
10	AR
11	SP

Control signals for IR register:

$L/\bar{H}$	Enable	FunSel	IR^+
Φ	0	Φ	IR
1	1	11	IR(0-7)←I
0	1	11	IR(8-15)←I
Φ	1	00	0
Φ	1	01	IR+1
Φ	1	10	IR-1

Selection signals for MUXA and MUXB:

MUXASel	MuxAOut	MUXBSel	MuxBOut
00	OutALU	00	OutALU
01	Address	01	Φ
10	Memory	10	Memory
11	IR(8-15)	11	IR(8-15)

Instruction format:

For instructions with address reference
IR(15-11): opcode
IR(10-9): regsel
IR(8): Addressing Mode
IR(7-0): Address

For instructions without address reference
IR(15-11): opcode
IR(10-8): destreg
IR(7-5): srcreg1
IR(4-2): srcreg2
IR(1-0): Not used

Instruction set:

OPCODE	SYMB	ADDR MODE	DESCRIPTION
0x00	LD	IM, D	Rx ← Value (Value is described in Table 3)
0x01	ST	D	Value ← Rx
0x02	MOV	N/A	DESTREG ← SRCREG1
0x03	PSH	N/A	M[SP] ← Rx, SP ← SP - 1
0x04	PUL	N/A	SP ← SP + 1, Rx ← M[SP]
0x05	ADD	N/A	DESTREG ← SRCREG1 + SRCREG2
0x06	SUB	N/A	DESTREG ← SRCREG2 - SRCREG1
0x07	DEC	N/A	DESTREG ← SRCREG1 - 1
0x08	INC	N/A	DESTREG ← SRCREG1 + 1
0x09	AND	N/A	DESTREG ← SRCREG1 AND SRCREG2
0x0A	OR	N/A	DESTREG ← SRCREG1 OR SRCREG2
0x0B	NOT	N/A	DESTREG ← NOT SRCREG1
0x0C	LSL	N/A	DESTREG ← LSL SRCREG1
0x0D	LSR	N/A	DESTREG ← LSR SRCREG1
0x0E	BRA	IM	PC ← Value
0x0F	BEQ	IM	IF Z=1 THEN PC ← Value
0x10	BNE	IM	IF Z=0 THEN PC ← Value
0x11	CALL	IM	M[SP] ← PC, SP ← SP – 1, PC ← Value
0x12	RET	N/A	SP ← SP + 1, PC ← M[SP]

Description of fields in the instruction set:

REGSEL	REGISTER
00	R0
01	R1
10	R2
11	R3

DESTREG/SRCREG1/SRCREG2	REGISTER
000	R0
001	R1
010	R2
011	R3
100	PC
101	PC
110	AR
111	SP

Addressing modes:

ADDRESSING MODE	MODE	SYMB	Value
0	Immediate	IM	ADDRESS Field
1	Direct	D	M[AR]

SOLUTION 3-b

T3D3: M[SP] 🡐 R0, SP 🡐 SP – 1, SC 🡐 0

M[SP] 🡐 R0:

- OutASel = 00
- FunSelALU = Transfer A
- OutCSel = 11
- MuxBSel = 00
- MemLoad = 1

SP 🡐 SP – 1

- FunSel = 10
- RegSel = 111

Control signals for register files:

FunSel	R_x^+
00	0
01	$R_x + 1$
10	$R_x - 1$
11	I (load)

OutA(B)Sel	OutA(B)	OutCSel	OutC
00	R0	00	PC
01	R1	01	PC
10	R2	10	AR
11	R3	11	SP

Control signals for IR register:

$L/\overline{H}$	Enable	FunSel	IR^+
Φ	0	Φ	IR
1	1	11	IR(0-7)←I
0	1	11	IR(8-15)←I
Φ	1	00	0
Φ	1	01	IR+1
Φ	1	10	IR-1

Selection signals for MUXA and MUXB:

MUXASel	MuxAOut	MUXBSel	MuxBOut
00	OutALU	00	OutALU
01	Address	01	Φ
10	Memory	10	Memory
11	IR(8-15)	11	IR(8-15)

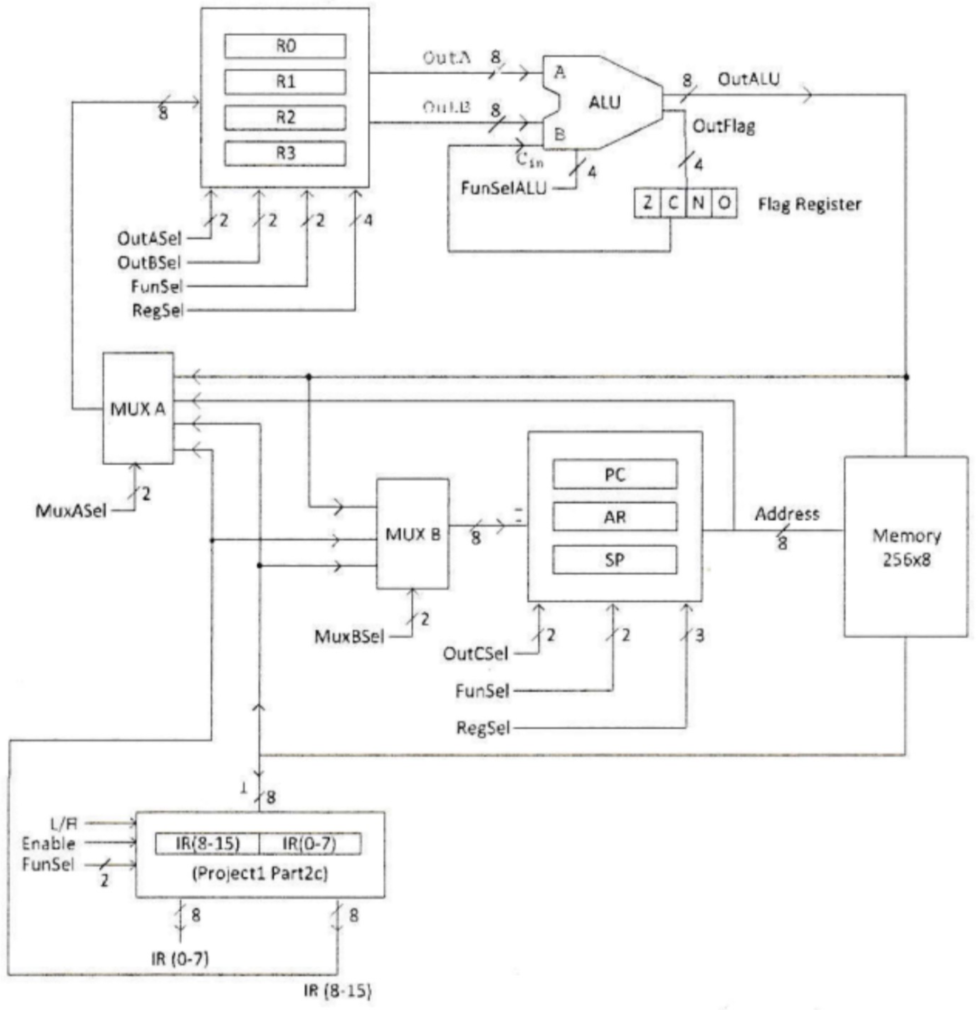
Description of fields in the instruction set:

REGSEL	REGISTER
00	R0
01	R1
10	R2
11	R3

DESTREG/SRCREG1/SRCREG2	REGISTER
000	R0
001	R1
010	R2
011	R3
100	PC
101	PC
110	AR
111	SP

Addressing modes:

ADDRESSING MODE	MODE	SYMB	Value
0	Immediate	IM	ADDRESS Field
1	Direct	D	M[AR]



QUESTION 4

PSH instruction decrements the value of the stack pointer (SP) by one, and then writes the content of DR into the memory cell whose address is pointed by SP.

PULL instruction loads a value into DR from the memory cell whose address is pointed by SP, and then increments the value of the stack pointer (SP) by one.

For Part D use the following information.

The microinstruction format of the software based control unit is as follows:

3	3	3	2	2	7
F1	F2	F3	CD	BR	AD

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

The microoperations for fields F1, F2, and F3 are given below:

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

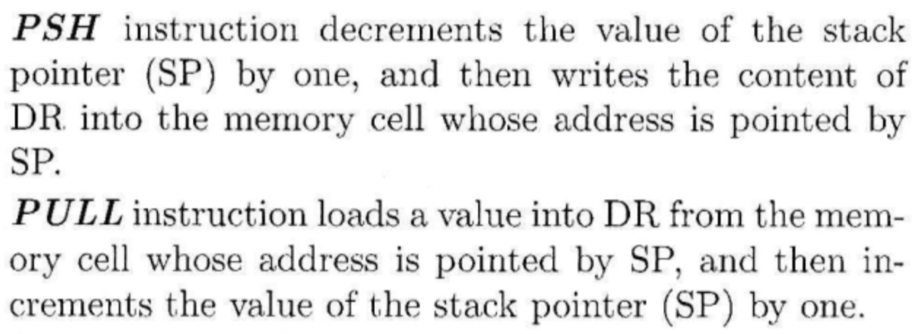
F2	Microoperation	Symbol
000	None	NOP
001	AC ← AC - DR	SUB
010	SP ← SP + 1	INCSP
011	SP ← SP - 1	DECSP
100	DR ← M[AR]	READ
101	DR ←AC	ACTDR
110	DR ←DR + 1	INCDR
111	DR(0-10) ← PC	PCTDR

F3	Microoperation	Symbol
000	None	NOP
001	AC ← AC ⊕ DR	XOR
010	AC ← AC'	COM
011	PC← SP	SP2PC
100	SP ← AR	ARTSP
101	PC ← PC + 1	INCPC
110	PC ← AR	ARTPC
111	AR ← SP	SPTAR

Write a symbolic microprogram for PSH (Opcode:101) and PULL (Opcode:110) using the information given. Beware of the changed mapping algorithm:
CAR[3 - 5] ← IR(12 - 14),CAR[0,1,2,6] ← 0

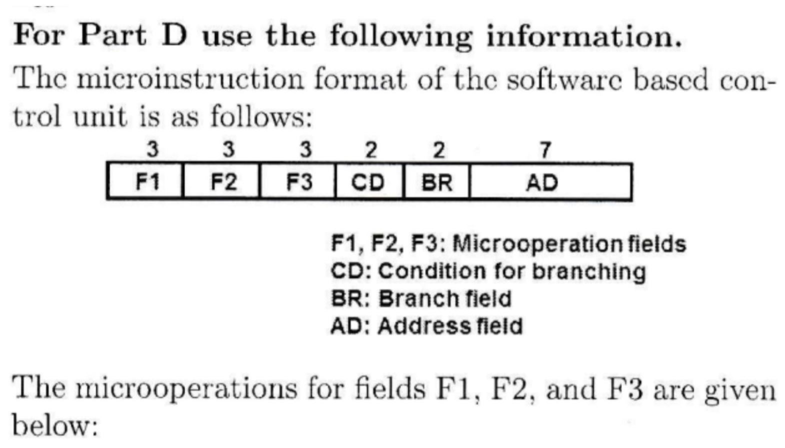
There is no need to implement fetch&decode cycles. Just assume that there is subroutine for fetch
(FETCH) available for your convenience that you can branch.

SOLUTION 4



PSH instruction decrements the value of the stack pointer (SP) by one, and then writes the content of DR into the memory cell whose address is pointed by SP.

PULL instruction loads a value into DR from the memory cell whose address is pointed by SP, and then increments the value of the stack pointer (SP) by one.



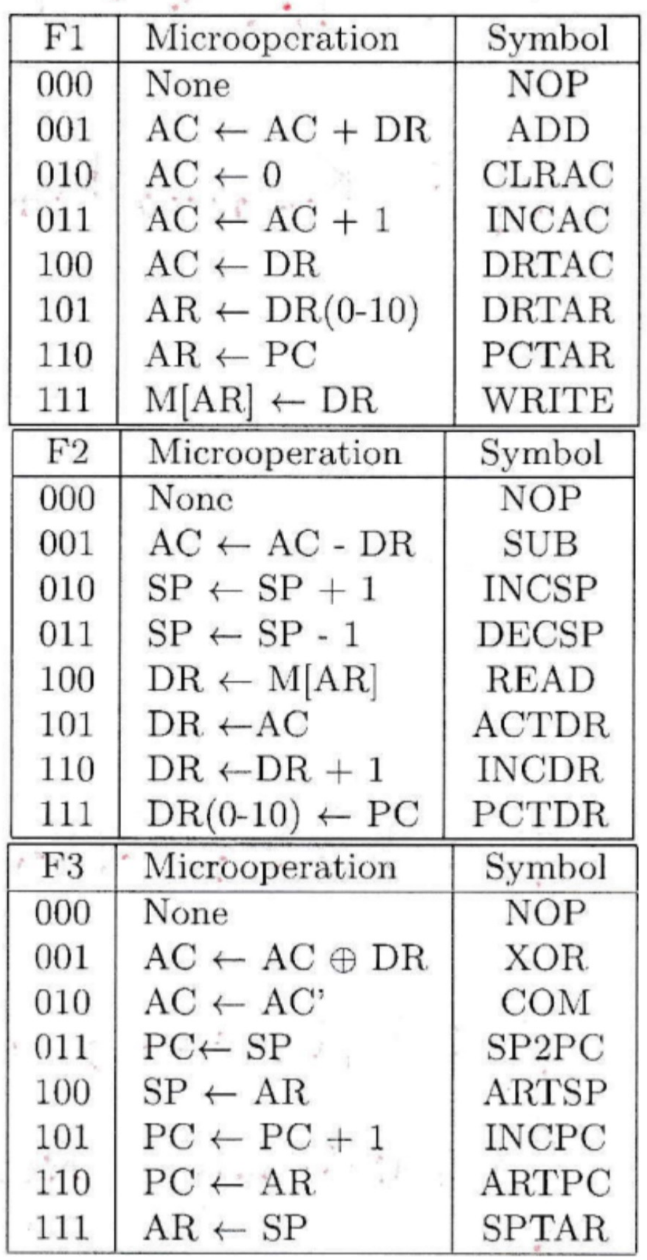
For Part D use the following information.

The microinstruction format of the software based control unit is as follows:

3	3	3	2	2	7
F1	F2	F3	CD	BR	AD

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

The microoperations for fields F1, F2, and F3 are given below:



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	SP ← SP + 1	INCSP
011	SP ← SP - 1	DECSP
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	PC← SP	SP2PC
100	SP ← AR	ARTSP
101	PC ← PC + 1	INCPC
110	PC ← AR	ARTPC
111	AR ← SP	SPTAR

CAR[3 - 5] ← IR(12 - 14),CAR[0,1,2,6] ← 0

PUSH: 0101000 = hex(28)

RTL:

T2K5: SP ← SP – 1
T3K5: AR ← SP
T4K5: M[AR] ← DR

Symbolic:

```
        ORG  0x28

PSH:    NOP, DECSP, NOP     U   J   NEXT
        NOP, NOP, SPTAR     U   J   NEXT
        WRITE, NOP, NOP     U   J   FETCH
```

SOLUTION 4

PSH instruction decrements the value of the stack pointer (SP) by one, and then writes the content of DR into the memory cell whose address is pointed by SP.

PULL instruction loads a value into DR from the memory cell whose address is pointed by SP, and then increments the value of the stack pointer (SP) by one.

For Part D use the following information.

The microinstruction format of the software based control unit is as follows:

3	3	3	2	2	7
F1	F2	F3	CD	BR	AD

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

The microoperations for fields F1, F2, and F3 are given below:

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	SP ← SP + 1	INCSP
011	SP ← SP - 1	DECSP
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	PC← SP	SP2PC
100	SP ← AR	ARTSP
101	PC ← PC + 1	INCPC
110	PC ← AR	ARTPC
111	AR ← SP	SPTAR

CAR[3 - 5] ← IR(12 - 14),CAR[0,1,2,6] ← 0

PULL: 0110000 = hex(30)

RTL:

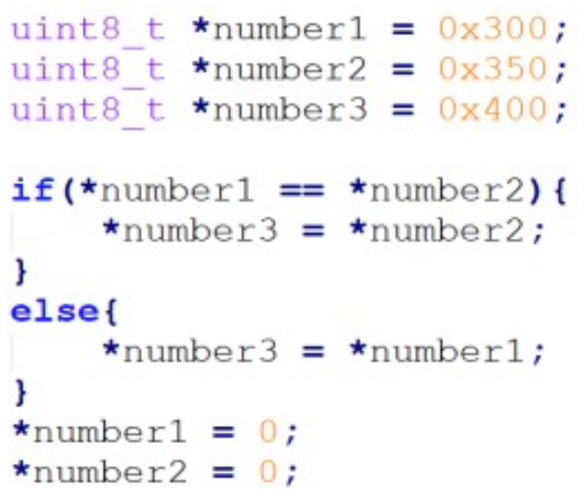
T2K6: AR ← SP
T3K6: DR ← M[AR], SP ← SP + 1,

Symbolic:

```
        ORG  0x30

PULL:   NOP,NOP, SPTAR      U   J   NEXT
        NOP, READ,NOP       U   J   NEXT
        NOP, INCSP, NOP     U   J   FETCH
```

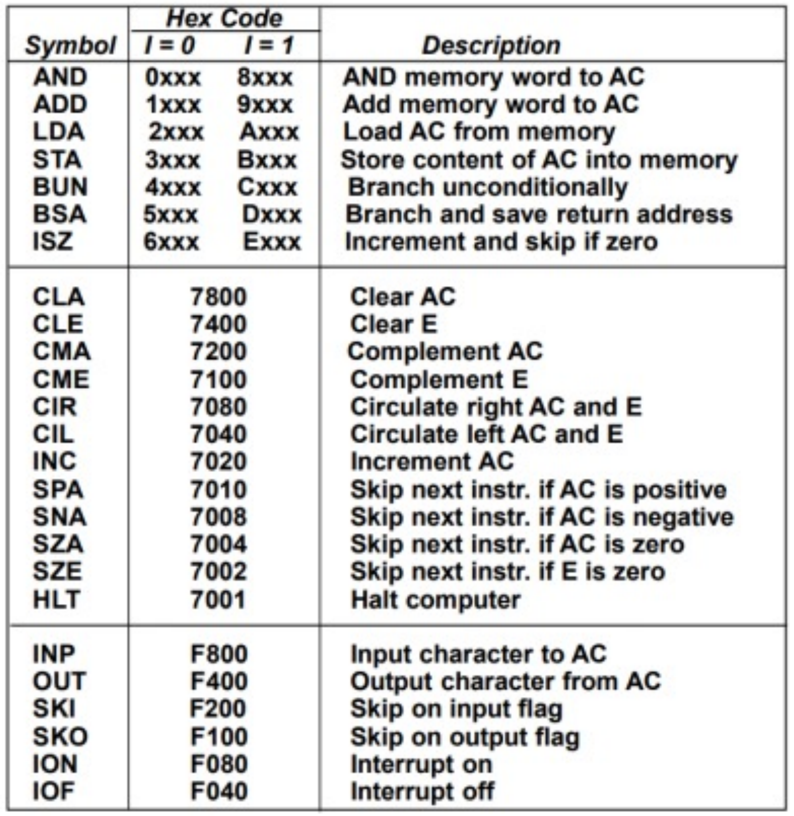
QUESTION 5

Write a program for Mano's Basic Computer to perform the same operations as done by the following C code. Specify all the hex codes of instructions. You must write both if and else blocks even if they do not execute. Assume that the program starts at 0x100. Note that uint8_t refers to eight-bit unsigned value.



```
uint8_t *number1 = 0x300;
uint8_t *number2 = 0x350;
uint8_t *number3 = 0x400;

if(*number1 == *number2){
    *number3 = *number2;
}
else{
    *number3 = *number1;
}
*number1 = 0;
*number2 = 0;
```

Table 1: Mano's Basic Computer Instructions



Symbol	Hex Code		Description
	I = 0	*I = 1*	
AND	0xxx	8xxx	AND memory word to AC
ADD	1xxx	9xxx	Add memory word to AC
LDA	2xxx	Axxx	Load AC from memory
STA	3xxx	Bxxx	Store content of AC into memory
BUN	4xxx	Cxxx	Branch unconditionally
BSA	5xxx	Dxxx	Branch and save return address
ISZ	6xxx	Exxx	Increment and skip if zero
CLA	7800		Clear AC
CLE	7400		Clear E
CMA	7200		Complement AC
CME	7100		Complement E
CIR	7080		Circulate right AC and E
CIL	7040		Circulate left AC and E
INC	7020		Increment AC
SPA	7010		Skip next instr. if AC is positive
SNA	7008		Skip next instr. if AC is negative
SZA	7004		Skip next instr. if AC is zero
SZE	7002		Skip next instr. if E is zero
HLT	7001		Halt computer
INP	F800		Input character to AC
OUT	F400		Output character from AC
SKI	F200		Skip on input flag
SKO	F100		Skip on output flag
ION	F080		Interrupt on
IOF	F040		Interrupt off

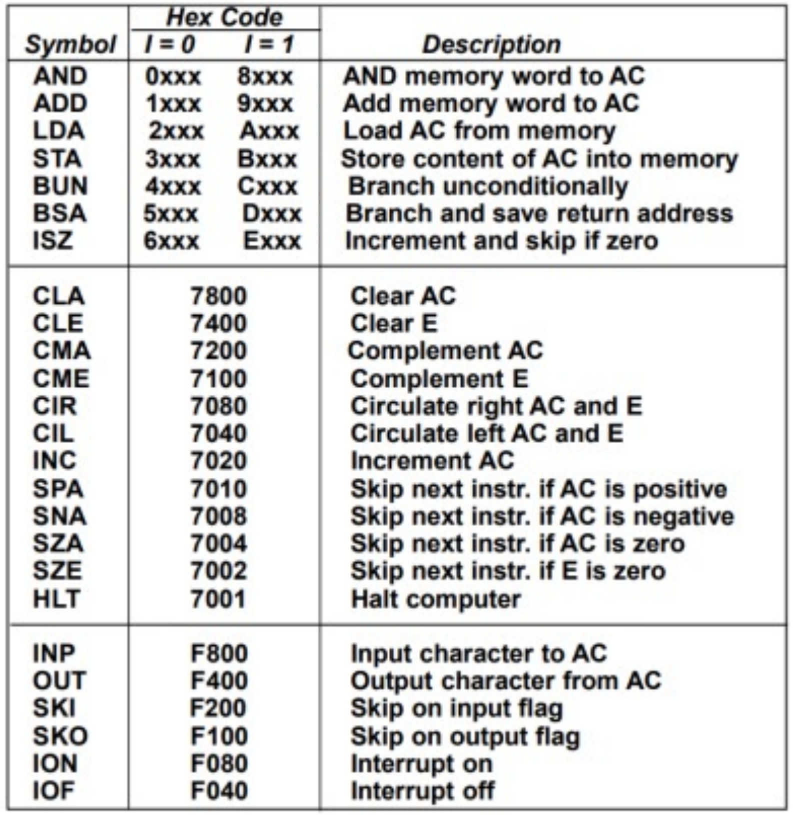
Table 2: Memory

Address	Content
0x300	5
:	:
0x350	8
:	:
0x400	4

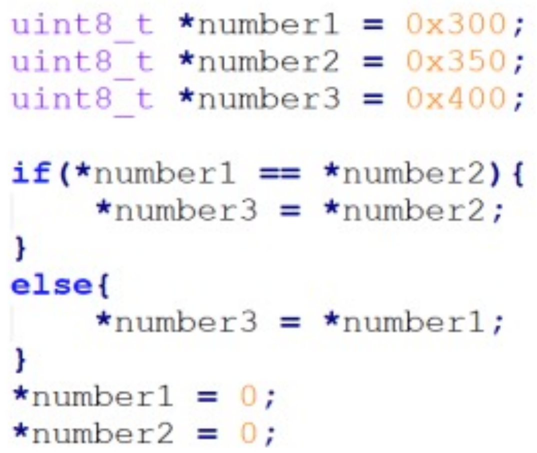
SOLUTION 5

```
ORG 100            ; Starting address of the program

; Load number1 (0x300) into AC
LDA 300            ; AC = M[300]
CMA                ; Complement AC (to perform subtraction)
INC                ; Increment AC (to perform 2's complement)
ADD 350            ; AC = AC + M[350]
SZA                ; Skip next instruction if AC is zero
BUN ELSE           ; Branch to ELSE if not equal

; IF BLOCK: *number3 = *number2
LDA 350            ; Load value at 0x350 into AC
STA 400            ; Store AC at 0x400
BUN ENDIF          ; Skip else block

; ELSE BLOCK
ELSE, LDA 300      ; Load value at 0x300 into AC
STA 400            ; Store AC at 0x400

; Set *number1 = 0
ENDIF, CLA         ; Clear AC
STA 300            ; Store 0 at 0x300

; Set *number2 = 0
STA 350            ; Store 0 at 0x350

HLT                ; Halt the program
```

Table 1: Mano's Basic Computer Instructions



Symbol	Hex Code I = 0	Hex Code I = 1	Description
AND	0xxx	8xxx	AND memory word to AC
ADD	1xxx	9xxx	Add memory word to AC
LDA	2xxx	Axxx	Load AC from memory
STA	3xxx	Bxxx	Store content of AC into memory
BUN	4xxx	Cxxx	Branch unconditionally
BSA	5xxx	Dxxx	Branch and save return address
ISZ	6xxx	Exxx	Increment and skip if zero
CLA	7800		Clear AC
CLE	7400		Clear E
CMA	7200		Complement AC
CME	7100		Complement E
CIR	7080		Circulate right AC and E
CIL	7040		Circulate left AC and E
INC	7020		Increment AC
SPA	7010		Skip next instr. if AC is positive
SNA	7008		Skip next instr. if AC is negative
SZA	7004		Skip next instr. if AC is zero
SZE	7002		Skip next instr. if E is zero
HLT	7001		Halt computer
INP	F800		Input character to AC
OUT	F400		Output character from AC
SKI	F200		Skip on input flag
SKO	F100		Skip on output flag
ION	F080		Interrupt on
IOF	F040		Interrupt off



```
uint8_t *number1 = 0x300;
uint8_t *number2 = 0x350;
uint8_t *number3 = 0x400;

if(*number1 == *number2){
    *number3 = *number2;
}
else{
    *number3 = *number1;
}
*number1 = 0;
*number2 = 0;
```

Table 2: Memory

Address	Content
0x300	5
:	:
0x350	8
:	:
0x400	4