Université d'Ottawa
Faculté de génie

École de science informatique
et de génie électrique

uOttawa
L'Université canadienne
Canada's university

University of Ottawa
Faculty of Engineering

School of Electrical Engineering
and Computer Science

CEG2136: Computer Architecture I / CEG2536: Architecture des Ordinateurs I

FINAL EXAMINATION - SAMPLES

A – SOLUTIONS ARE HIGHLIGHTED

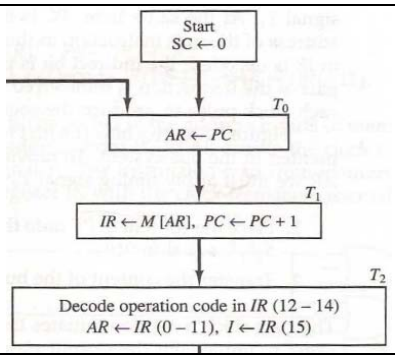
Q1.1 For 1 K memory locations you need:

(a) 8 address lines
(b) 10 address lines
(c) 12 address lines
(d) None of the above

(A memory with a capacity of 2^m words requires m address lines. So, for $2^m = 2^{10}$, we need $m = 10$ address lines)

Q1.2 Which CPU register provides the address from which the next instruction opcode is to be fetched?

(a) Instruction register IR
(b) Accumulator AC
(c) Program counter PC
(d) None of these

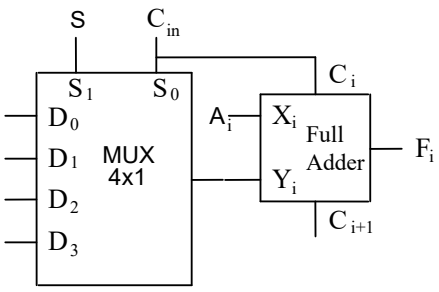


Q1.4 An arithmetic unit performs basic operations on two numbers A and B, under the control of two bits S and C_{in}., as shown in the following table:

S	$C_{in} = 0$	$C_{in} = 1$
0	F = A + B (addition)	F = A + 1 (increment A)
1	F = A - 1 (decrement A)	F = A + B'+ 1 (subtraction)

The following combinational circuit (multiplexor and full adder) is used to implement the functions described above for bit *i*. Select from the following combinations of multiplexor inputs which set of logic values implements correctly the above functions

S	Cin = 0	Cin = 1
0	$F = A_2 A_1 A_0 + B_2 B_1 B_0$	$F = A_2 A_1 A_0 +$ 0 0 0 + 1
1	$F = A_2 A_1 A_0 +$ 1 1 1	$F = A_2 A_1 A_0 + B'_2 B'_1 B'_0 + 1$

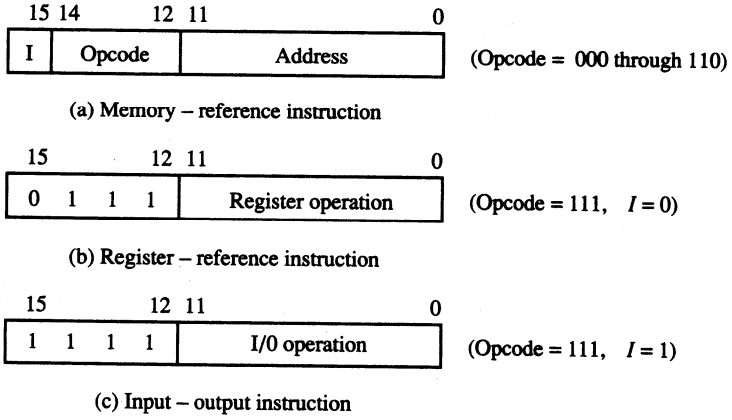


(a) $D_0=B_i'$, $D_1=0$, $D_2=1$, $D_3=B_i$
(b) $D_0=0$, $D_1=B_i$, $D_2=B_i'$, $D_3=1$
(c) $D_0=1$, $D_1=B_i'$, $D_2=B_i$, $D_3=0$
(d) $D_0=B_i$, $D_1=0$, $D_2=1$, $D_3=B_i$'
(e) $D_0=1$, $D_1=B_i$, $D_2=B_i'$, $D_3=0$

S C_{in}	X_i	Y_i
00	A_i	B_i
01	A_i	0
10	A_i	1
11	A_i	B'i

Q2

The block diagram of the basic computer that was introduced in chapter 5 of Mano's textbook is given in the annex, along with its instruction list. The instruction word is 16 bit long and has the following structure …



Q2.1 At some point, the content of **PC** of the basic computer is **3AF** (all numbers are in hexadecimal) and the content of **AC** is **2EC3**, as shown in the following table. The content of memory is partially given below, as well:

MEMORY

Address	Memory content
3AD	03B5
3AE	ABBA
3AF	93AD
3B0	DEED
3B1	7BEE
3B2	AD08
3B3	10BC
3B4	1CAA
3B5	3B9F
3B6	3BA0

BASIC COMPUTER REGISTERS

	Content before instruction execution	Content after instruction execution
PC	**3AF**	3B0
AC	**2EC3**	6A62
AR	0000	3B5
DR	0000	3B9F
IR	0000	93AD
E	0	0
I	0	1
SC		000

a. What is the instruction that will be fetched and executed?

ADD 3AD I

b. Show the operands and the binary operation that will be performed in the *AC* when the instruction is executed.

2EC3 + 3B9F

c. Fill out the last column ("Content after instruction execution") of the above table with the contents of registers PC, AR, DR, AC, and IR in hexadecimal and the values of E, I and the sequence counter SC in binary, all shown at the end of the instruction cycle.

Q2.2 What is the result, in decimal, of the operation performed by the following assembly program?

		Instruction effect
	ORG 100	
	LDA OP	AC=FFA5
	CMA	AC=005A
	INC	AC=005B
	ADD OP1	AC=0094
	STA OP2	M(OP2)=**0094**
	HLT	
OP1,	0039	
OP,	FFA5	
OP2,	0	**0094**
	END	

HEX94 = DEC(9 x 16 + 4) = 144 + 4 = 148

The machine code of this program is stored in a memory unit of 1 kilo-word of 16 bits implemented on an Altera FPGA; give the Quartus .mif file that describes this program

Assembly Program		Memory Address	Memory content
	ORG 100		
	LDA OP	100	2107
	CMA	101	7200
	INC	102	7020
	ADD OP1	103	1106
	STA OP2	104	3108
	HLT	105	7001
OP1,	HEX 39	106	0039
OP,	- HEX 5B	107	FFA5
OP2,	0	108	0000
	END		

Quartus .mif file:

Addr	+0	+1	+2	+3	+4	+5	+6	+7
100	2107	7200	7020	1106	3108	7001	0039	FFA5
108	0000	0000	0000	0000	0000	0000	0000	0000
110	0000	0000	0000	0000	0000	0000	0000	0000

Q4.1 Examining the annexed block diagram of the Mano's basic computer, give the list of the blocks of the datapath and the list of those that form the computer's control unit.

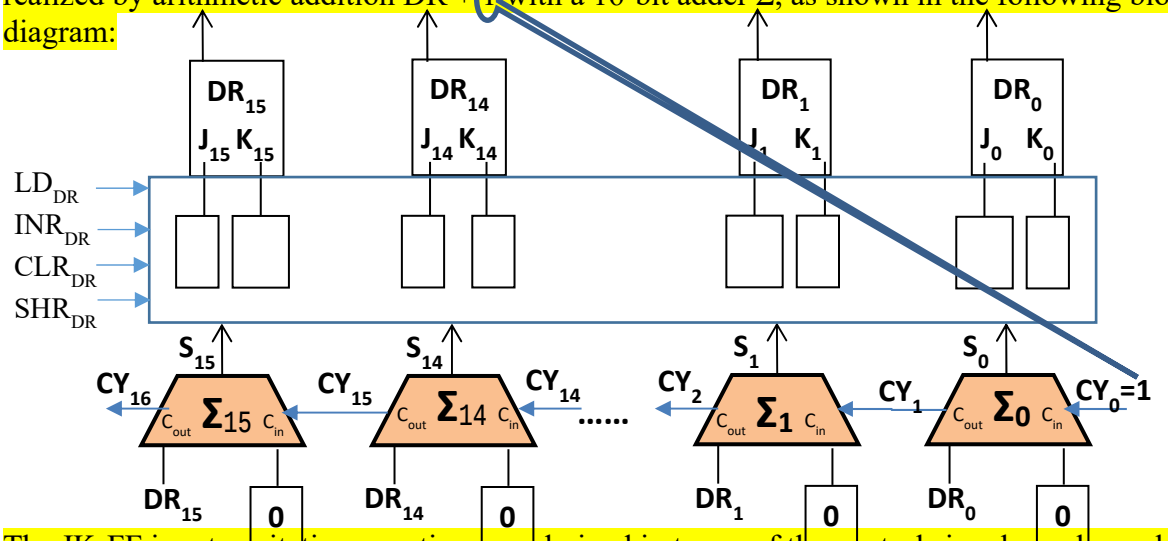
Q4.2 You have to expand the instruction list of your textbook basic computer with three more instructions (ASR, DIV, TAT) as shown in the following table**;** both circuits of the datapath and the control unit need to be extended. This basic computer is a simplified version of the one presented in the textbook as it does not have any provisions to deal with interrupts.

The datapath has to be redesigned to allow your DR register to support the following operations:

	RTL	/ Comments
LD_{DR} = x T_4:	DR ← (bus)	/Transfer value from bus to DR
INR_{DR} = y T_3 :	DR ← DR+1	/Increment DR
CLR_{DR} = z T_3:	DR ← 0	/Clear DR
SHR_{DR} = v T_3 + w T_4:	DR ← shr DR	/Shift right DR

Design and draw the logic diagram of DR_i (bit *i* of register DR), using any type of combinational circuits and a JK flip-flop.

Solution: DR will be implemented as a ***multi-function register*** where incrementation INR_{DR} is realized by arithmetic addition DR + 1 with a 16-bit adder Σ, as shown in the following block diagram:



The JK-FF input excitation equations are derived in terms of the control signals as shown below:

J^n	K^n	Q^{n+1}	*Function*	*Action:*
0	0	Q^n	Hold	To keep the current state, do not include any term in J & K equations
0	1	0	Reset	To reset a JK-FF, add that term to the K equation, but not to J
1	0	1	Set	To set a JK-FF, add that term to the J equation, but not to K
1	1	Q^n	Toggle	To toggle a JK-FF, add that term to both J & K equations

To load value *a* into a JK-FF, AND *a* with the term and include it in the J equation, while *a*' is AND-ed with the term and it is applied to K. In particular, to load whatever next state, set $a = Q^+$ (JK-FF as D-FF)

4.2.a Applying these rules to our problem, the equations of the J and K inputs can be written directly:

$J_{DRi} = xT_4\, bus_i + yT_3\, S_i + (vT_3 + wT_4)\, DR_{i+1}$ for i = 0…14,

$K_{DRi} = xT_4\, bus'_i + yT_3 S'_i + (vT_3 + wT_4) DR'_{i+1} + zT_3$

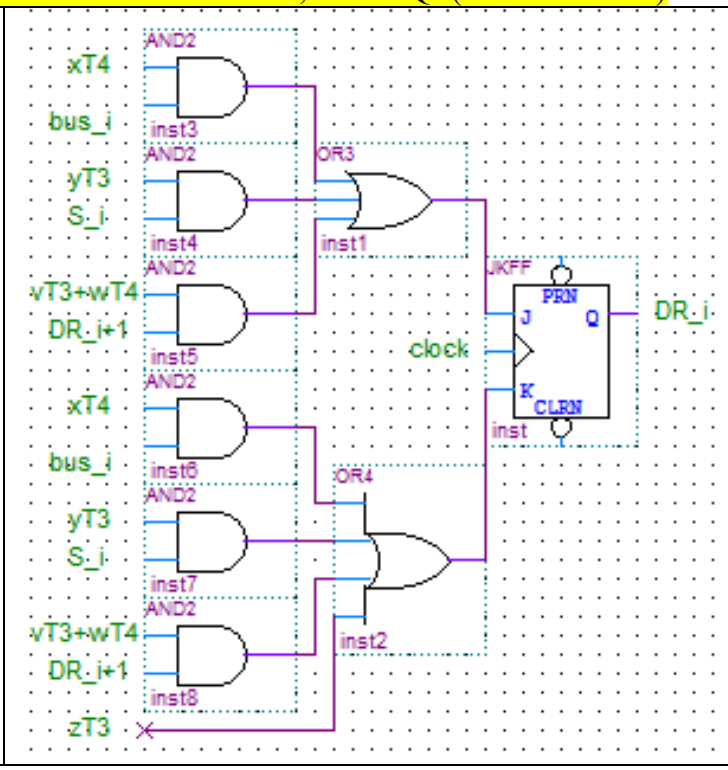
and, for **arithmetic** shift right, DR_{15} has to be copied to both most significant two bits, so,

$J_{DR15} = x\, T_4\, bus_{15} + y\, T_3\, S_{15} + (v\, T_3 + w\, T_4)\, DR_{15}$

$K_{DR15} = xT_4\, bus'_{15} + yT_3\, S_{15}' + (vT_3 + wT_4) DR'_{15} + zT_3$

Alternatively, the above equations can be derived from the following MEV truth table:

RTL		J_{DRi}	K_{DRi}
Condition	Micro-operation		
$LD_{DR} = x\, T_4$:	DR ← (bus)	bus_i	bus'_i
$INR_{DR} = y\, T_3$:	DR←DR+1	S_i	S'_i
$CLR_{DR} = z\, T_3$:	DR ← 0	0	1
$SHR_{DR} = vT_3 + wT_4$:	DR←shr DR	DR_{i+1}	DR'_{i+1}



4.2.b Alternatively, the initial table is re-written as shown below:

Function	Present state	Next state	JK FF inputs		JK-FF
RTL	DR_i	DR_i^+	J_{DR}	K_{DR}	Implemented as
LD_{DR} = x T_4: DR ← (bus)	0	bus_i	**bus_i**	***d*** (bus_i')	D-FF
	1	bus_i	***d*** (bus_i)	**bus_i'**	
INR_{DR} = y T_3 : DR ← DR+1 (implemented here with DR ← S/**Σ**)	0	S_i	**S_i**	***d*** (S'_i)	D-FF
	1	S_i	***d*** (S_i)	**(S_i)'**	
CLR_{DR} = z T_3: DR ← 0	0	0	0	***d*** (1)	JK Reset
	1	0	***d*** (0)	1	
SHR_{DR} = v T_3 + w T_4: DR ← shr DR	0	DR_{i+1}	**DR_{i+1}**	***d*** (DR'_{i+1})	D-FF
	1	DR_{i+1}	***d*** (DR_{i+1})	**DR'_{i+1}**	

Don't care is marked in the above table with "***d***"

DR_i variable can be eliminated from J and K equations if the inputs are made the same for both DR_i = 0 or 1 as shown in the parentheses following the ***d*** 's in the table; i.e., the JK-FF's emulates D-FF.

$J_{DRi} = xT_4\, bus_i + yT_3\, S_i + (vT_3 + wT_4)\, DR_{i+1}$

$K_{DRi} = xT_4\, bus'_i + yT_3\, S'_i + (vT_3 + wT_4)\, DR'_{i+1} + zT_3$

for i = 0…14 and

$J_{DR15} = x\, T_4\, bus_{15} + y\, T_3\, S_{15} + (v\, T_3 + w\, T_4)\, DR_{15}$

$K_{DR15} = xT_4\, bus'_{15} + y\, T_3\, S_{15}' + (v\, T_3 + w\, T_4)\, DR'_{15} + zT_3$

Since the second Σ operand is 0, the same can be realized by using half-adders instead of full-adders Σ:

$S_j = DR_j \oplus CY_{j-1}$;

$CY_j = DR_j \bullet CY_{j-1}$

with j = 1…15 and

$S_0 = DR_0 \oplus 1$; $CY_0 = DR_0 \bullet 1$

4.2.c Another alternate can employ a comprehensive table, as follows:

Inputs							Present state	Next state		
xT_4	yT_3	zT_3	$vT_3 + wT_4$	bus_i	S_i	DR_{i+1}	DR_i	DR_i^+	J_i	K_i
LD_{DR}	INR_{DR}	CLR_{DR}	SHR_{DR}							
1	x	x	x	0	x	x	0	0	0	x
1	x	x	x	0	x	x	1	0	x	1
1	x	x	x	1	x	x	0	1	1	x
1	x	x	x	1	x	x	1	1	x	0
x	**1**	x	x	x	0	x	0	0	0	x
x	**1**	x	x	x	0	x	1	0	x	1
x	**1**	x	x	x	1	x	0	1	1	x
x	**1**	x	x	x	1	x	1	1	x	0
x	x	1	x	x	x	x	0	0	0	x
x	x	1	x	x	x	x	1	0	x	1
x	x	x	1	x	x	0	0	0	0	x
x	x	x	1	x	x	0	1	0	x	1
x	x	x	1	x	x	1	0	1	1	x
x	x	x	1	x	x	1	1	1	x	0

$J_i = xT_4\, bus_i + yT_3\, S_i + (vT_3 + wT_4)\, DR_{i+1}$

$K_i = xT_4\, bus'_i + yT_3\, S'_i + zT_3 + (vT_3 + wT_4)\, DR'_{i+1}$

Σ with Half-adder:

$S_i = DR_i \oplus CY_{i-1}$

$CY_i = DR_i \bullet CY_{i-1}$

Q5

Q5.1 RTL notation is used in the following table to describe the FETCH cycle of a memory reference instruction. Give short explanations of the respective micro-operations in the last column

State	RTL Micro-operations	Explanations
T_0	$AR \leftarrow PC$	Address of the instruction to be executed is loaded into the address register
T_1	$IR \leftarrow M[AR]$ $PC \leftarrow PC + 1$	The instruction is loaded in the Instruction Register Address of next instruction is prepared
T_2	$D_0, D_1, ..., D_7 \leftarrow DecodeIR(14-12)$ $AR \leftarrow IR(11-0)$ $I \leftarrow IR(15)$	Opcode is decoded Address of operand (if direct addressing) or pointer to the address (if indirect addressing) goes to AR Direct/indirect bit goes to bit I
$I \cdot D_7^{'} \cdot T_3$	$AR \leftarrow M[AR]$	

Q5.2a Use RTL notation to describe the EXECUTION cycle of each of the three newly added instructions; give the corresponding control signals in terms of the instruction code and the sequence counter. Use as many lines for your table as you need; if needed more, please use the back of the page.

ASR F020	**Arithmetic Right Shift (DR ← DR/2)**
DIV F010	**Divide by 4 (DR ← DR/4)**
TAT F008	**Swap AC with DR (DR ←→ AC)**

Instr.	RTL Condition	RTL Micro-operation	Control signals: Bus Select S_2, S_1, S_0	SHR_{DR}	LD_{DR}	LD_{AC}	INR_{SC}	CLR_{SC}	ALU_{DR}
ASR	D_7 I T_3 IR_5:	DR ← shr DR SC ← 0	000	1	0	0	0	1	0
TAT	D_7 I T_3 IR_3:	AC <= DR, DR <=AC SC ← 0	100	0	1	1	0	1	1
DIV	D_7 I T_3 IR_4:	DR ← shr DR SC ← SC+1	000	1	0	0	1	0	0
	D_7 I T_4 IR_4:	DR← shr DR SC ← 0	000	1	0	0	0	1	0

Q5.2b Derive the equations of the control signals from the above table:

$SHR_{DR} = D_7 I T_3 IR_5 + D_7 I T_3 IR_4 + D_7 I T_4 IR_4$

$LD_{DR} = D_7 I T_3 IR_3 = LD_{AC} = ALU_{DR}$

$INR_{SC} = D_7 I T_3 IR_4$

$CLR_{SC} = D_7 I T_3 IR_5 + D_7 I T_3 IR_3 + D_7 I T_4 IR_4$

$S_2 = D_7 I T_3 IR_3$

$S_1 = 0$

$S_0 = 0$

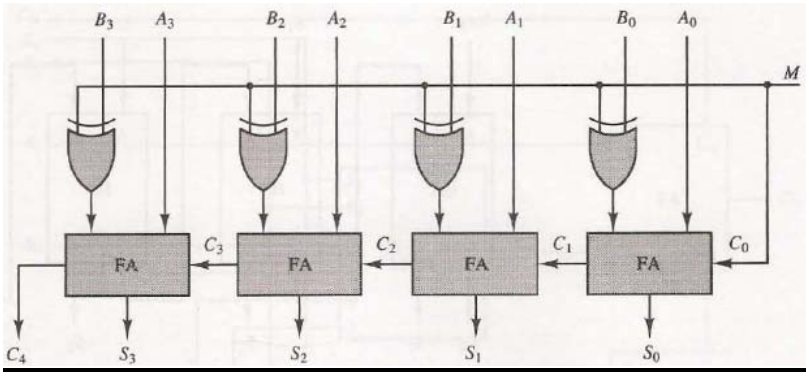
NOTE: The term D_7 I T_3 IR_x is obtained from the instruction in the instruction register IR as follows:

ASR	F	0	2	0
IR	1111	0000	0010	0000
Decoder	I D_7		IR_5	
TAT	F	0	1	0
IR	1111	0000	0001	0000
Decoder	I D_7		IR_4	
DIV	F	0	0	8
IR	1111	0000	0000	1000
Decoder	I D_7			IR_3

B

Question 1

Consider the following logic diagram:



a) What is the C_4, S_3, S_2, S_1, S_0 value if:

(1) Register A = 1101, Register B = 0110, and M=0

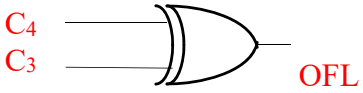
(2) Register A = 1101, Register B = 0110, and M=1

Write your calculations and results in the last two columns of the following table

b) Given that registers A, B and S contain signed numbers in **2's complement representation**, convert the binary numbers A_3 A_2 A_1 A_0, B_3 B_2 B_1 B_0 and S_3 S_2 S_1 S_0 to decimal, and write your results and calculations in the following table:

A_3 A_2 A_1 A_0 (binary/ decimal)	$B_3B_2B_1B_0$ (binary/ decimal)	M	C_4	a) S_3 S_2 S_1 S_0 (binary)	b) S_3 S_2 S_1 S_0 (decimal)	c) Are the results provided by the logic circuit correct?, i.e., are they in accord with the results you obtain if you do the same operations yourself
A=1101 = -|A| |A|=-1101= 2's compl(1101) |A|=0011 = 3 A = -3	0110 = + 6	0	1	Carry: $c_4c_3c_2c_1c_0$ 1 1 0 0 0 1101 +0110 =(1)0011	S = A + B S = - 3 + 6 =0011=+3	CORRECT!
1101	0110 = + 6	1	1	Carry : 1 0 011 1101 +1001 (1)0111	S = A – B S = -3 – 6 = 0111= +7 !?	A -B = -3 – 6 = -9 => Overflow, since the result should be in the domain [-8, +7)

d) Expand your adder with a circuit that can signalize overflow



Question 3

Consider the computer of Lab 4, the architecture of which is described in Figure 1 and Tables 1, 2, 3, 4, and 5. The instruction type is determined by the two most significant bits of the 8-bit register IR, as follows:
X_0 = IR(7)' IR(6)' denotes a memory-reference instruction (MRI) in direct addressing mode;
X1 = IR(7)' IR(6) denotes a register-reference instruction (RRI); and
X_2 = IR(7) IR(6)' denotes a memory-reference instruction (MRI) in indirect addressing mode.

Assume that all registers are equipped with 3 control bits for loading the register (LD), increment by 1 (INC), and reset to zero (CLR).

Find the list of all micro-operations which change the value of register AC and derive the logic equations.

MRI

	Notation (RTL)
Y0 = IR(6)' IR(1) IR(0)'	T6 Y0:DR ← M [AR] T7 Y0:AC← AC + DR SC← 0
Y1 = IR(6)' IR(2)	T6 Y1:DR ← M [AR] T7 Y1:AC← DR, SC← 0
Y2 = IR(6)' IR(3)	T6: T7 Y2:M [AR] ← AC, SC←0
Y3 = IR(6)' IR(4)	T6 Y3:PC← AR , SC← 0
Y4 = IR(6)' IR(5)	T6 Y4:DR ← M [AR] T7 Y4:DR ← DR + 1 T8 Y4:M [AR] ← DR T_9 Y_4:if (DR= 0)S' then (PC← PC+ 1) $T_{10}Y_4$:if (DR = 0)S' then (PC← PC+ 1) T10 Y4:SC← 0
Y5 = IR(6)'IR(1)'IR(0)	T6 Y5:DR ← M [AR] T7 Y5:AC← AC ∧ DR SC← 0
Y6 = IR(6)' IR(1)'IR(0)	T6 Y6:DR ← M [AR] T7 Y6:AC← AC − DR SC← 0

RRI

Symbol	Notation (RTL)
CLA	T3 X1 IR(0): AC ← 0
CMA	T3 X1 IR(1): AC ← AC
ASL	T3 X1 IR(2): AC ← ashl AC
ASR	T3 X1 IR(3): AC ← ashr AC
INC	T3 X1 IR(4): AC ← AC + 1
HLT	T3 X1 IR(5) : S ← 1

T7 Y0 : AC ← AC + DR ,
T7 Y1 : AC ← DR
T7 Y5 : AC ← AC ∧ DR
T7 Y6 : AC ← AC – DR

T3 X1 IR(0) : AC ← 0
T3 X1 IR(1) : AC ← AC
T3 X1 IR(2) : AC ← ashl AC
T3 X1 IR(3) : AC ← ashr AC
T3 X1 IR(4) : AC ← AC + 1

LD_{AC} = T7 (Y0 + Y1 + Y5 + Y6) + T3[(X1 IR(1) + X1 IR(2) + X1 IR(3)]

INC_{AC} = T3 X1 IR(4)

CLR_{AC} = T3 X1 IR(0)

Question 4

Two memory-reference instructions in the Basic Computer are to be changed to the instructions specified in the following table.

Symbol	Opcode	Symbolic designation	Description in words
ADM	001	$M[EA] \leftarrow AC + M[EA]$	Add AC to memory
BNA	011	If $(AC < 0)$ then $(PC \leftarrow EA)$	Branch if AC is negative

In the following table, write down the necessary micro-operations (in RTL) to perform the execution phase of each instruction. Note that the execution phase of such type of instructions starts at T_4. See figures 2 and 3 from Annex. **Note** that the value in AC should not be changed by the execution of any instruction unless the instruction specifies a change in its content. To this extent you can use TR to store the content of AC temporary.

Question 5

Column (3) of the following table shows the content of a memory segment of the Basic Computer that stores a machine language program and its operands.

1. Convert from binary to hexadecimal both the address and the memory contents (columns (1) and (3)) and write your results in columns (2) and (4), respectively.
2. Use Table 6 from Annex to convert the machine language program (stored in the first six memory locations) to symbolic operation codes (assembly language). Use Table 6 from Annex to convert

   the machine language program (stored in the first six memory locations) to symbolic operation codes (assembly language). Write the assembly language instructions in the corresponding cells of column (5) of the above table. Use hexadecimal representation for the address field of the instructions.
3. Fill out the last column (6) of the table with the content of the accumulator after the execution of each instruction.

Symbol	RTL
ADM	D_1T_4: $TR \leftarrow AC$ D_1T_5: $DR \leftarrow M[EA]$ D_1T_6: $AC \leftarrow AC \mid DR$ D_1T_7: $M[EA] \leftarrow AC$ D_1T_8: $DR \leftarrow TR$ D_1T_9: $AC \leftarrow DR$, $SC \leftarrow 0$
BNA	D_3T_4: If $(AC(15) = 1)$ then $(PC \leftarrow AR)$, $SC \leftarrow 0$

Memory Address		Memory Content		Assembly /operands in HEX	**AC** after execution of instr.
Binary (1)	Hex (2)	Binary (3)	Hex (4)	(5)	(6)
1000 0001 0111	817	0111 1000 0000 0000	7800	CLA	0000
1000 0001 1000	818	0010 1000 0001 1101	281D	LDA 81D	0121
1000 0001 1001	819	0001 1000 0001 1110	181E	ADD 81E	0130
1000 0001 1010	81A	1000 1000 0001 1111	881F	AND 81F I	0100
1000 0001 1011	81B	0011 1000 0001 1101	381D	STA 81D	0100
1000 0001 1100	81C	0111 0000 0000 0001	7001	HLT	0100
1000 0001 1101	81D	0000 0001 0010 0001	0121		
1000 0001 1110	81E	0000 0000 0000 1111	000F		
1000 0001 1111	81F	0000 1000 0010 0000	0820		
1000 0010 0000	820	0000 0001 1100 0100	01C4		

The instruction list

Symbol	Hex code
AND	0 or 8
ADD	1 or 9
LDA	2 or A
STA	3 or B
BUN	4 or C
BSA	5 or D
ISZ	6 or B
CLA	7800
CLE	7400
CMA	7200
CME	7100

Symbol	Hex code
CIR	7080
CIL	7040
INC	7020
SPA	7010
SNA	7008
SZA	7004
SZE	7002
HLT	7001
INP	F800
OUT	F400
SKI	F200

Question 6

a) Write a subroutine (MUL) in assembly language that multiplies two positive numbers by a repeated addition method. For example, to multiply 6 x 4, the subroutine evaluates the product by adding the multiplicand (6) four times (multiplier), i.e., 6+6+6+6.
Write your program using instructions of the Basic Computer given in the Table 6 from the Annex. Your subroutine should be stored in the memory of your Basic Computer beginning with address **A00**.

Before invoking your subroutine MUL, the calling program places:

- the multiplicand (6 in the above example) in memory location at address **9FF**;
- the multiplier (4 in the above example) in the accumulator.

The result (the product of the two numbers) is passed by subroutine to the calling program through the accumulator.
NOTE: You are allowed to use different ways to pass parameters, but full explanations should be provided.

Solution:

```
        ORG HEX 9FF
MLP,    0000            / a_i
MUL,    0000
        CMA
        INC             /initialize CTR to
        STA CTR /"negative multiplier" -b_i
        CLA
LOP,    ADD MLP
        ISZ CTR
        BUN LOP
        BUN MUL I
CTR,    0000            / - b_i
```

b) This part is independent of part (a).
The coordinates of two 2-dimensional vectors (a_x, a_y, b_x, b_y) are stored at consecutive addresses starting at F01. Write an assembly language program that calculates the scalar (dot) product of two 2-dimensional vectors, assuming that all their coordinates are positive numbers:

$$p = a_x b_x + a_y b_y$$

Your program is stored in memory beginning with location at address **100,** and it has to place the product "p" at address **F00**.

To calculate partial products ($a_x b_x$ and $a_y b_y$), you should call subroutine MUL. Parameters have to be passed by "call-by-value," as follows:

- memory location **9FF** is used to pass a factor, say a_x;
- the other factor (i.e., b_x) is passed to the subroutine through the accumulator;
- the product $a_x b_x$ is passed back to your program by subroutine MUL through the accumulator.

Solution:

```
        ORG HEX 100
        CLA             /to initialise P = 0
        STA P
        LDA AX /prepare a_x in MLP
        STA MLP / pass the first factor a_x
        LDA BX /prepare b_x in AC to pass it to MUL
        BSA MUL     / call MUL to get a_x b_x
        ADD P       / P ← P + a_x b_x = 0+ a_x b_x
        STA P
        LDA AY /prepare a_y in MLP
        STA MLP / pass the first factor a_y
        LDA BY      /prepare b_y in AC
        BSA MUL     / call MUL to get a_y b_y
        ADD P  / P <- P + a_y b_y = a_x b_x + a_y b_y
        STA P
        HLT
        ORG HEX F00
P
AX              / a_x
AY              / a_y
BX              / b_x
BY              / b_y
```

The following is an alternate solution where "parameter linkage" is used instead of passing the multiplicand through the global variable MLP at 9FF; still "call-by-value" is used here.

```
        ORG HEX 100  / MAIN program calculates ax bx + ay by
        CLA          / to initialise
        STA P        /   P = 0
        LDA AX/prepare multiplicand ax in ML1
        STA ML1 / pass the first factor / multiplicand ax
        LDA BX / prepare multiplier bx in AC to pass it to MUL
        BSA MUL      / call MUL to get ax bx
ML1                  / ax
        ADD P       / P ← P + ax bx = 0 + ax bx
        STA P
        LDA AY   / prepare multiplicand ay in ML2
        STA ML2 / pass the first factor / multiplicand ay
        LDA BY       /prepare multiplier by in AC
        BSA MUL      / call MUL to get ay by
ML2                  / ay
        ADD P  / P ← P + ay by = ax bx + ay by
        STA P
        HLT
```

```
        ORG HEX F00
P
AX              / ax
AY              / ay
BX              / bx
BY              / by
```

```
        ORG HEX A00       /    Subroutine MUL
MUL,    0000
        CMA         / negate multiplier
        INC         / and
        STA CTR / initialize CTR to "negative multiplier"
        CLA    / Initialize the sum of AC = AC + MLi x CTR
LOP,    ADD MUL I    / use parameter linkage to add multiplicand
                       to itself as many times as the multiplier
        ISZ CTR
        BUN LOP
        ISZ MUL / sync MUL to get correct return address
        BUN MUL I / return from subroutine
CTR,    0000
```

ANNEX

The 8-bit Computer of Lab 4

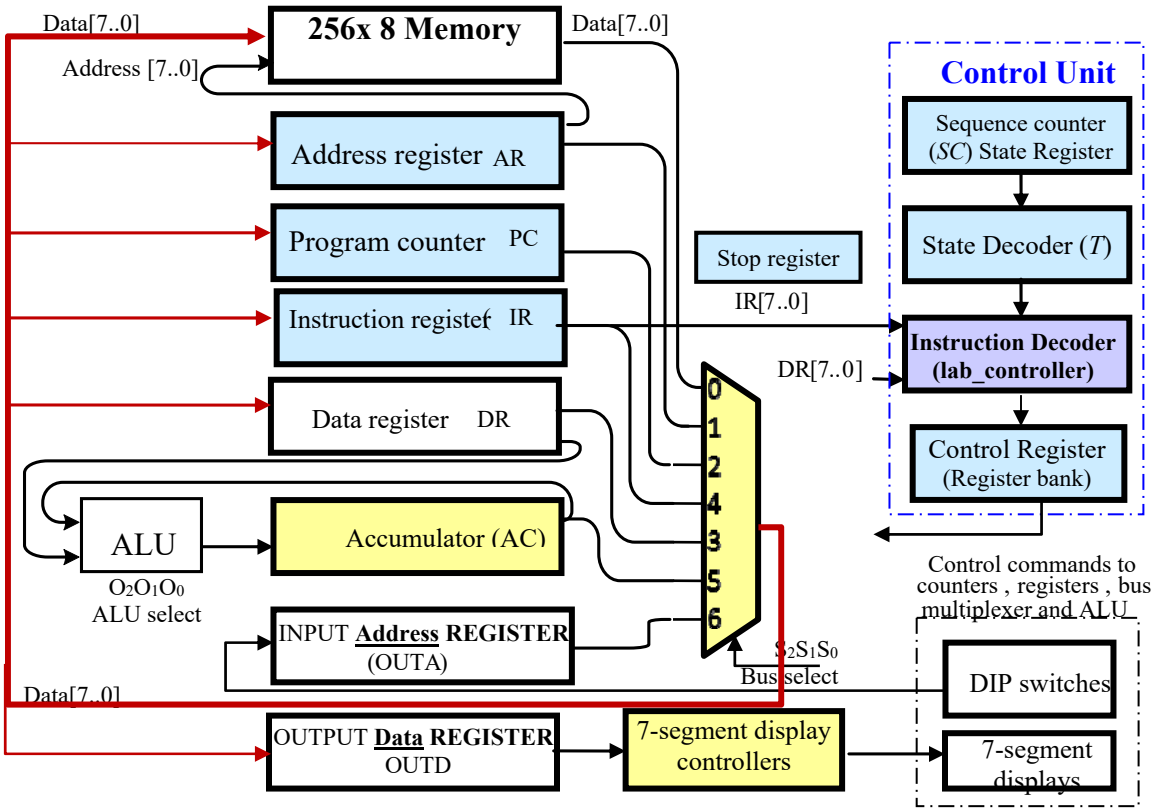


Fig. 1. Block diagram of the 8-bit Computer of Lab 4

Table 1: Starting micro-operations in the instruction cycle of the 8-bit mini computer

Instant	Description	Notation (RTL)
T0	Load the program counter PC in AR, and increment PC	T0 : AR ← PC T0 S′ : PC ← PC + 1
T1	Read instruction from memory and put it in the instruction register IR	T1 : IR ← M [AR]
T2	This cycle is not used to allow for the new value of the instruction to propagate in the controller	(nothing)
T3	If X_1 => it is a *register - reference instruction* that will be executed now (RRI)	T3 X1: execute an RRI instruction (Table 4) T3 X1 : SC ← 0
	If X_0 or X_2 , read the memory address of the operand and place it in AR, and increment PC. Recall that (X_0 + X_2) = IR(6)'	T3 IR(6)' : AR ← PC T3 IR(6)' S': PC ← PC + 1
T4	Read memory address into AR	T4 IR(6)' : AR ← M [AR]
T5	If *indirect addressing*, read the *operand address* from memory location pointed to by AR	T5 X2 : AR ← M [AR]
	If *direct addressing*, don't do anything, as the operand address is already in AR since T_6	T5 X0 : (nothing)
starting from T6	Execute the memory-reference instructions (MRI) described in Table 2	(see Table 2)

Table 2: Execution of a memory-reference instruction (MRI) in the 8-bit mini computer

Symbol		Notation (RTL)
ADD	Y0 = IR(6)' IR(1) IR(0)'	T6 Y0 : DR ← M [AR] T7 Y0 : AC ← AC + DR , SC ← 0
LDA	Y1 = IR(6)' IR(2)	T6 Y1 : DR ← M [AR] T7 Y1 : AC ← DR , SC ← 0
STA	Y2 = IR(6)' IR(3)	T6 : (cycle not used to allow for the address bus to stabilize) T7 Y2 : M [AR] ← AC , SC ← 0
BUN	Y3 = IR(6)' IR(4)	T6 Y3 : PC ← AR , SC ← 0
ISZ (assuming that the next instruction is a memory-reference instruction, stored at 2 memory locations further down)	Y4 = IR(6)' IR(5)	T6 Y4 : DR ← M [AR] T7 Y4 : DR ← DR + 1 T8 Y4 : M [AR] ← DR T_9 Y_4 : if (DR = 0)S' then (PC ← PC + 1) $T_{10}Y_4$: if (DR = 0)S' then (PC ← PC + 1) T10 Y4 : SC ← 0
AND	Y5 = IR(6)' IR(1)' IR(0)	T6 Y5 : DR ← M [AR] T7 Y5 : AC ← AC ∧ DR , SC ← 0
SUB	Y6 = IR(6)' IR(1)' IR(0)	T6 Y6 : DR ← M [AR] T7 Y6 : AC ← AC − DR , SC ← 0

Table 3: Execution of register-reference instructions (RRI) of the 8-bit mini computer

Symbol	Notation (RTL)
CLA	T_3 X_1 IR(0): AC ← 0
CMA	T_3 X_1 IR(1) : AC ← AC'
ASL	T_3 X_1 IR(2): AC ← ashl
ASR	T_3 X_1 IR(3): AC ← ashr
INC	T_3 X_1 IR(4): AC ←AC+1
HLT	T_3 X_1 IR(5): S ← 1

Table 4:
Function table of the 8-bit ALU

$O_2O_1O_0$	ALU operation
000	F = AC + DR / addition
001	F = AC – DR / subtraction
010	F = ashl AC / arithmetic shift left
011	F = ashr AC / arithmetic shift right
100	F = AC ∧ DR / AND
101	F = AC ∨ DR / OR
110	F = AC' / 1's complement
111	F = DR / transfer DR to AC

Table 5:
Function table of the 8-bit bus

S2	S1	S0	Register placed on the bus
0	0	0	Memory
0	0	1	AR
0	1	0	PC
0	1	1	DR
1	0	0	IR
1	0	1	AC
1	1	0	OUTA
1	1	1	None

The 16-bit Basic Computer of the textbook / L'ordinateur de base 16 bits du manuel

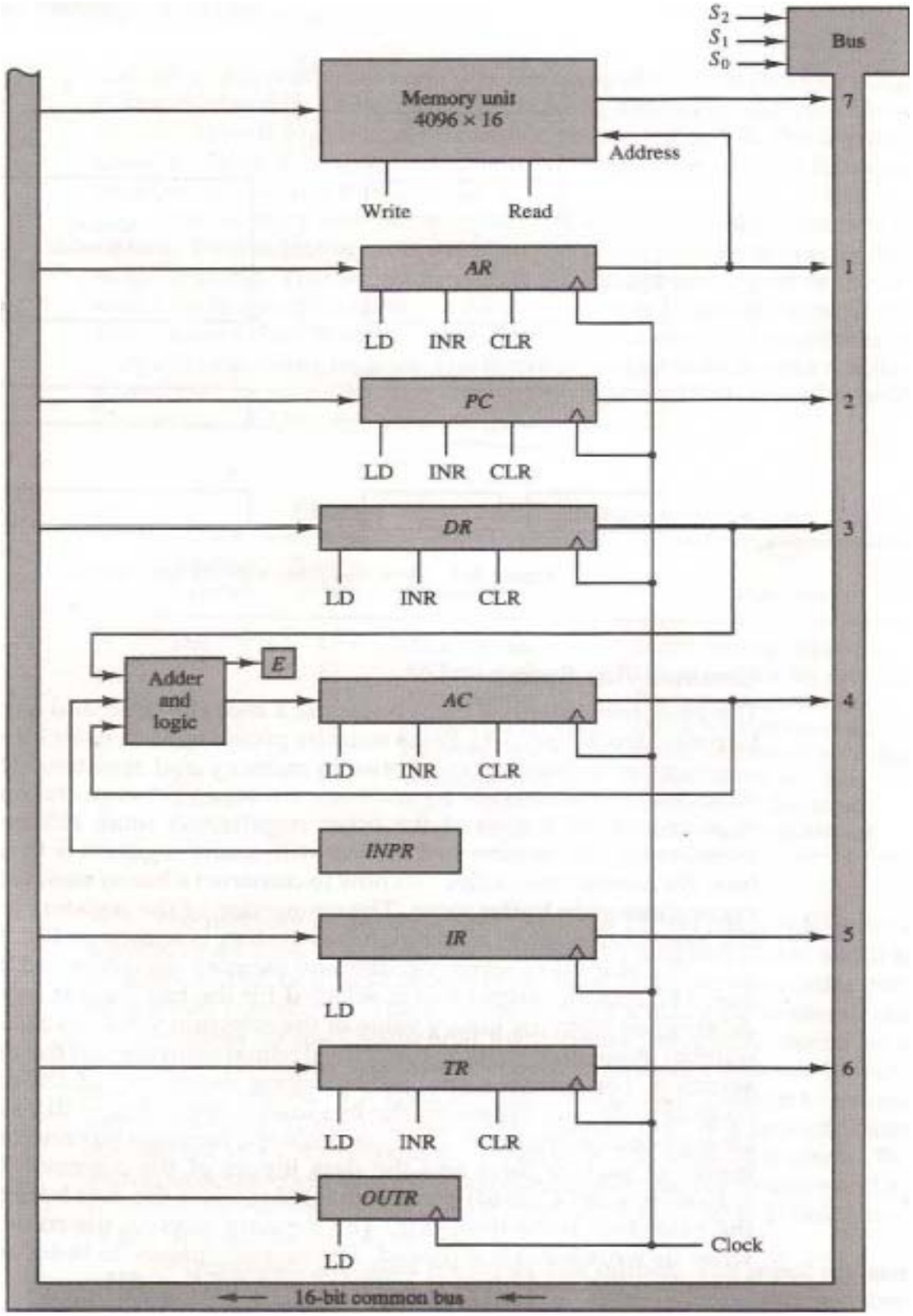


Fig. 2. Basic Computer registers connected to a common bus

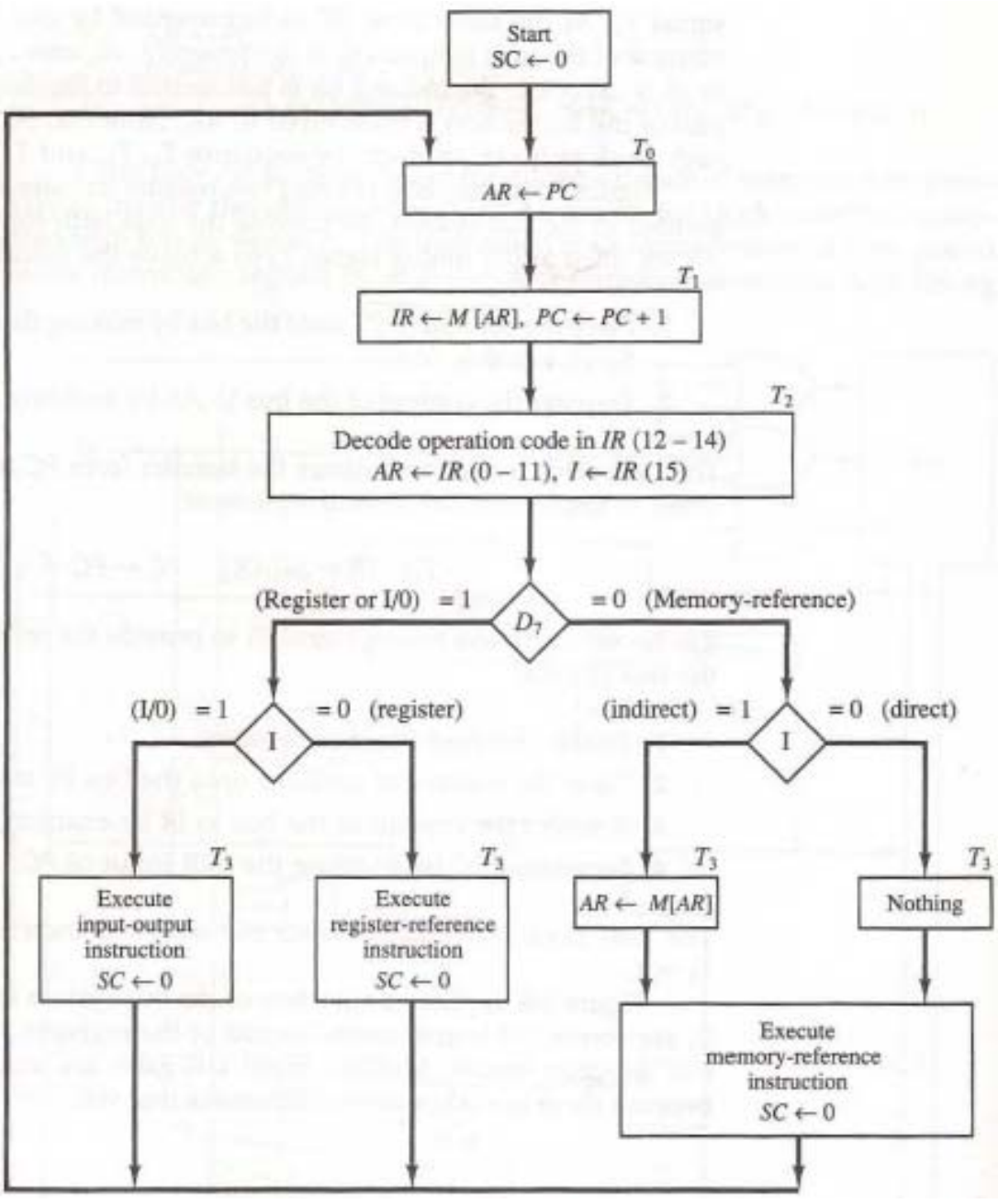


Fig. 3. Principle flowchart for instruction cycle of the 16-bit **Basic Computer** of the textbook

Table 6.

Basic Computer Instructions List

Symbol	Hex 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,AC*
CIL	7040	Circulate left *E, AC*
INC	7020	Increment *AC,*
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

Table 7. **RTLs of the Basic Computer**

Fetch	$R'T_0$:	
	$R'T_1$:	
Decode	$R'T_2$:	
Indirect	D'_7IT_3:	$AR \leftarrow M[AR]$
Interrupt:	$T_0'T_1'T_2'$	$IEN(FGI+FGO)$: $R \leftarrow 1$
	RT_0:	$AR \leftarrow 0$, $TR \leftarrow PC$
	RT_1:	$M[AR] \leftarrow TR$, $PC \leftarrow 0$
	RT_2:	$PC \leftarrow PC+1$, $IEN \leftarrow 0$, $R \leftarrow 0$, $SC \leftarrow 0$
Memory - reference instructions		
AND	D_0T_4:	$DR \leftarrow M[AR]$
	D_0T_5:	$AC \leftarrow AC \wedge DR$, $SC \leftarrow 0$
ADD	D_1T_4:	$DR \leftarrow M[AR]$
	D_1T_5:	$AC \leftarrow AC+DR$, $E \leftarrow C_{out}$, $SC \leftarrow 0$
LDA	D_2T_4:	$DR \leftarrow M[AR]$
	D_2T_5:	$AC \leftarrow DR$, $SC \leftarrow 0$
STA	D_3T_4:	$M[AR] \leftarrow AC$, $SC \leftarrow 0$
BUN	D_4T_4:	$PC \leftarrow AR$, $SC \leftarrow 0$
BSA	D_5T_4:	$M[AR] \leftarrow PC$, $AR \leftarrow AR + 1$
	D_5T_5:	$PC \leftarrow AR$, $SC \leftarrow 0$
ISZ	D_6T_4:	$DR \leftarrow M[AR]$
	D_6T_5:	$DR \leftarrow DR + 1$
	D_6T_6:	$M[AR] \leftarrow DR$, $SC \leftarrow 0$
	D_6T_6DR':	$PC \leftarrow PC + 1$
Register - reference instructions		
	$D_7I'T_3 = r$	(common to all RRI)
	$IR(i) = B_i$	(i = 0, 1, 2, ... , 11)
	r:	SC $\leftarrow 0$
CLA	rB_{11}:	$AC \leftarrow 0$
CLE	rB_{10}:	$E \leftarrow 0$
CMA	rB_9:	$AC \leftarrow AC'$
CME	rB_8:	$E \leftarrow E'$
CIR	rB_7:	$AC \leftarrow shr\ AC$, $AC(15) \leftarrow E$, $E \leftarrow AC(0)$
CIL	rB_6:	$AC \leftarrow shlAC$, $AC(0) \leftarrow E$, $E \leftarrow AC(15)$
INC	rB_5:	$AC \leftarrow AC + 1$
SPA	$rB_4AC'(15)$:	$PC \leftarrow PC + 1$
SNA	rB_3AC(15):	$PC \leftarrow PC + 1$
SZA	rB_2AC':	$PC \leftarrow PC + 1$
SZE	rB_1E':	$PC \leftarrow PC + 1$
HLT	rB_0:	$S \leftarrow 0$
Input - output instructions		
	$D_7IT_3 = p$	(common to all IOI)
	$IR(i) = B_i$	(i = 6, 7, 8, 9, 10, 11)
	p:	$SC \leftarrow 0$
INP	pB_{11}:	$AC(0\text{-}7) \leftarrow INPR$, $FGI \leftarrow 0$
OUT	pB_{10}:	$OUTR \leftarrow AC(0\text{-}7)$, $FGO \leftarrow 0$
SKI	pB_9FGI:	$PC \leftarrow PC + 1$
SKO	pB_8FGO:	$PC \leftarrow PC + 1$
ION	pB_7:	$IEN \leftarrow 1$
IOF	pB_6:	$IEN \leftarrow 0$