

CONTROLLED BY MAIN SWITCH N°1

240V G.P.O.

240V G.P.O.

240V MAIN SWITCH N°1

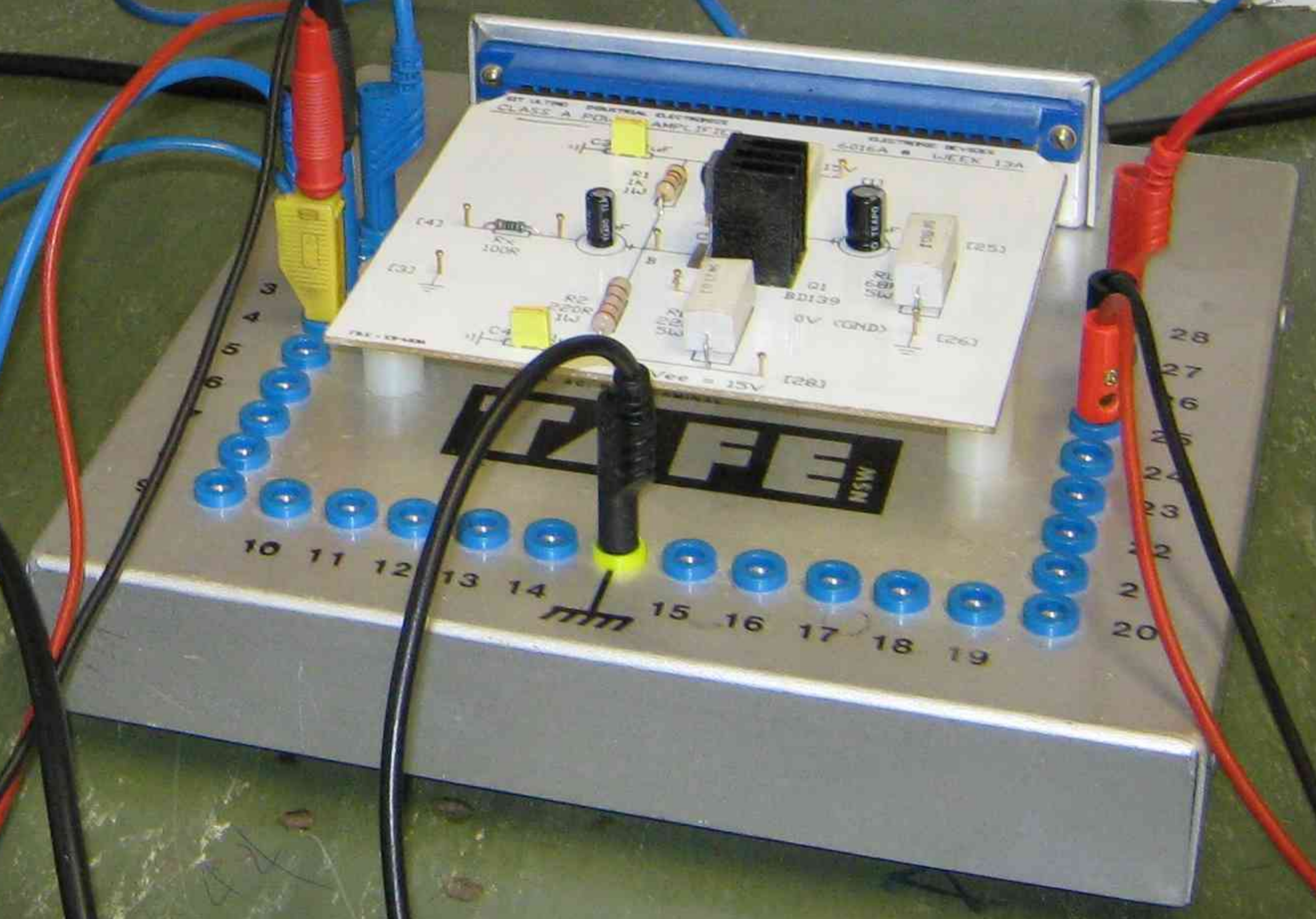
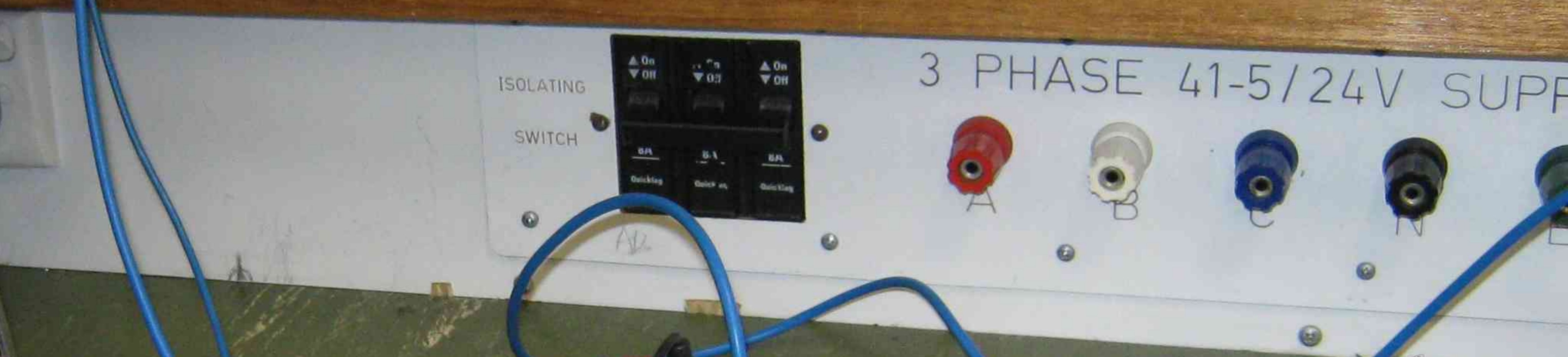
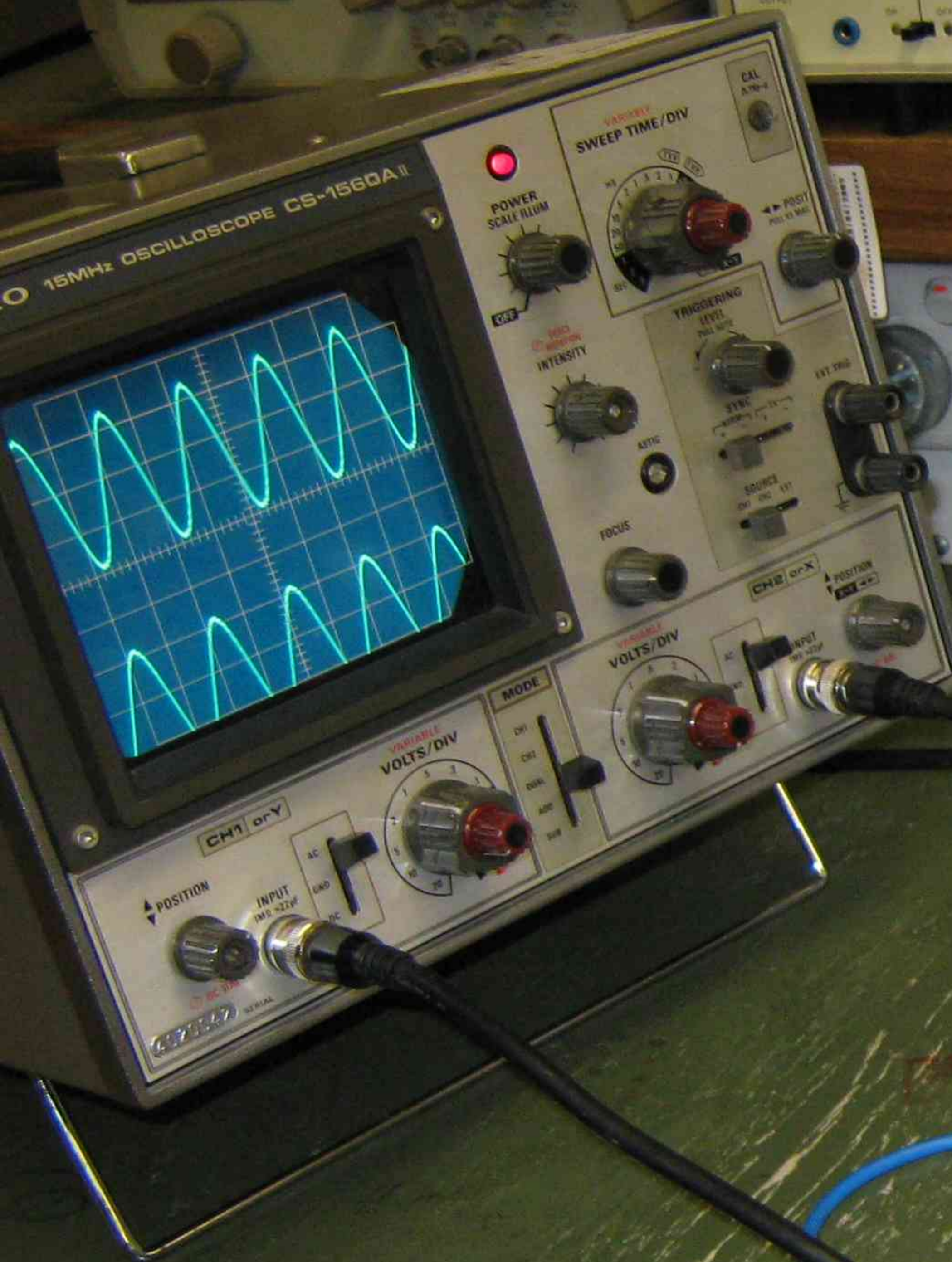
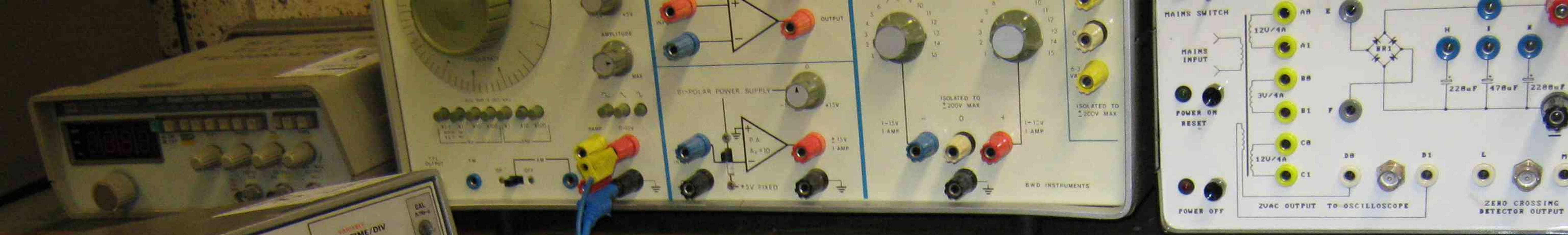
NEUTRA

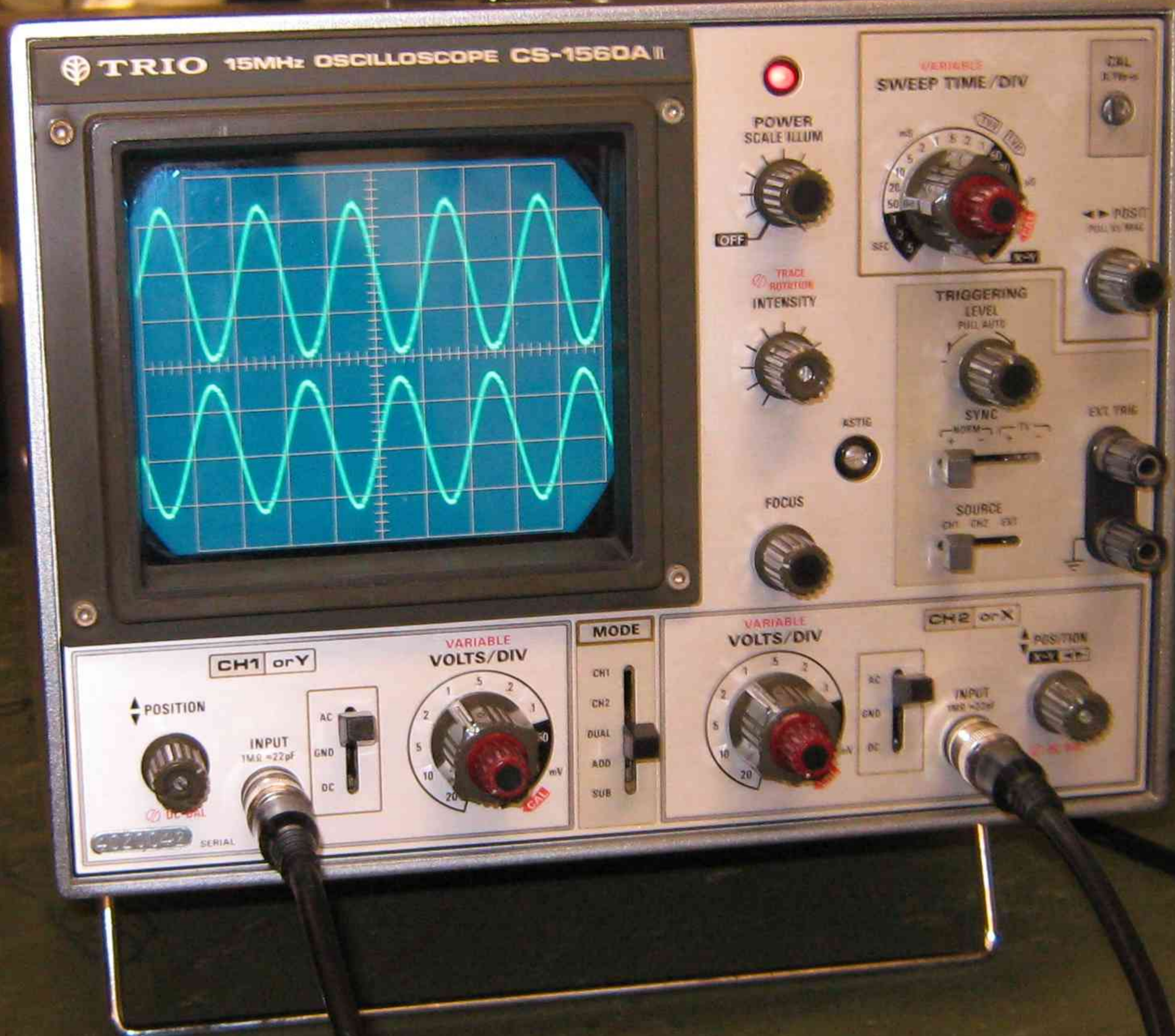
EARTH

240V MAIN SWITCH N°2

240V G.P.O.

ANY RUBBISH ON THE... PLEASE PUT IT IN THE... OF THE...





Put equipment away
Hang leads neat and tidy

MINI-LAB MODEL 603B

FUNCTION GENERATOR

VOLTAGE/OPERATIONAL AMPLIFIER

REGULATED POWER SUPPLY

LOW VOLTAGE AC/DC PWR SUPPLY

SYDNEY INSTITUTE OF TECHNOLOGY

MAIN SWITCH

POWER ON RESET

POWER OFF

DUAL TRACKING DC POWER SUPPLY

MODEL 20-2-E

CURRENT SLAVE VOLTAGE

NORMAL TRACKING

COMMON W TRACKING

FUNCTION GENERATOR

DIGITAL DISPLAY

TRIO 5000 OSCILLOSCOPE

TRIGGER

MODE

SCALE

3 PHASE 41-5/24V SUPPLY

ISOLATING SWITCH

A B C N E

3 PHASE 41-5/24V SUPPLY

Put equipment away
Hang leads neat and tidy

TRIO 15MHz OSCILLOSCOPE CS-1560A1

POWER SCALE ILLUM
TRIGGERING LEVEL
SOURCE CH1 CH2 EXT
CH1 or Y VOLTS/DIV
CH2 or X VOLTS/DIV
SWEEP TIME/DIV

FUNCTION GENERATOR

MINI-LAB MODEL 6038L BWD

VOLTS/DIV
BI-POLAR POWER SUPPLY
REGULATED POWER SUPPLY

LOW VOLTAGE AC/DC PWR SUPPLY

SYDNEY INSTITUTE OF TECHNOLOGY

12V/4A
30V/4A
12V/4A

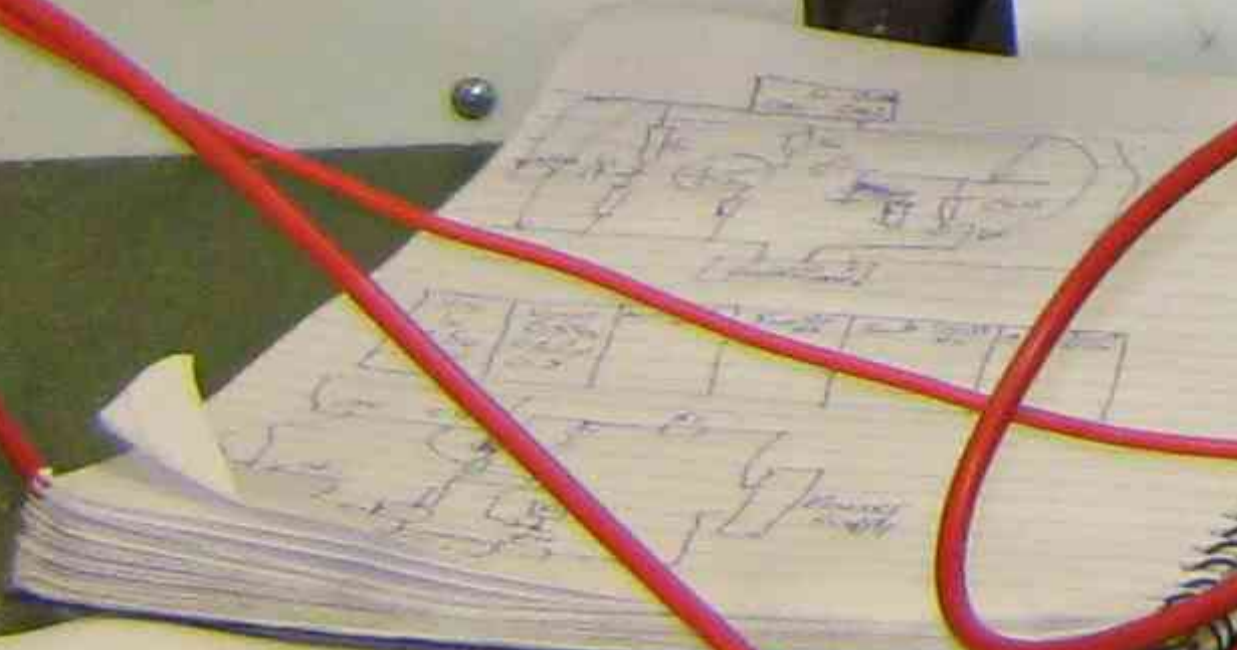
FUNCTION GENERATOR

3W

3.1818

H324

TELETYPE



Class (A) power amplifier
Have (D) sketch the circuit

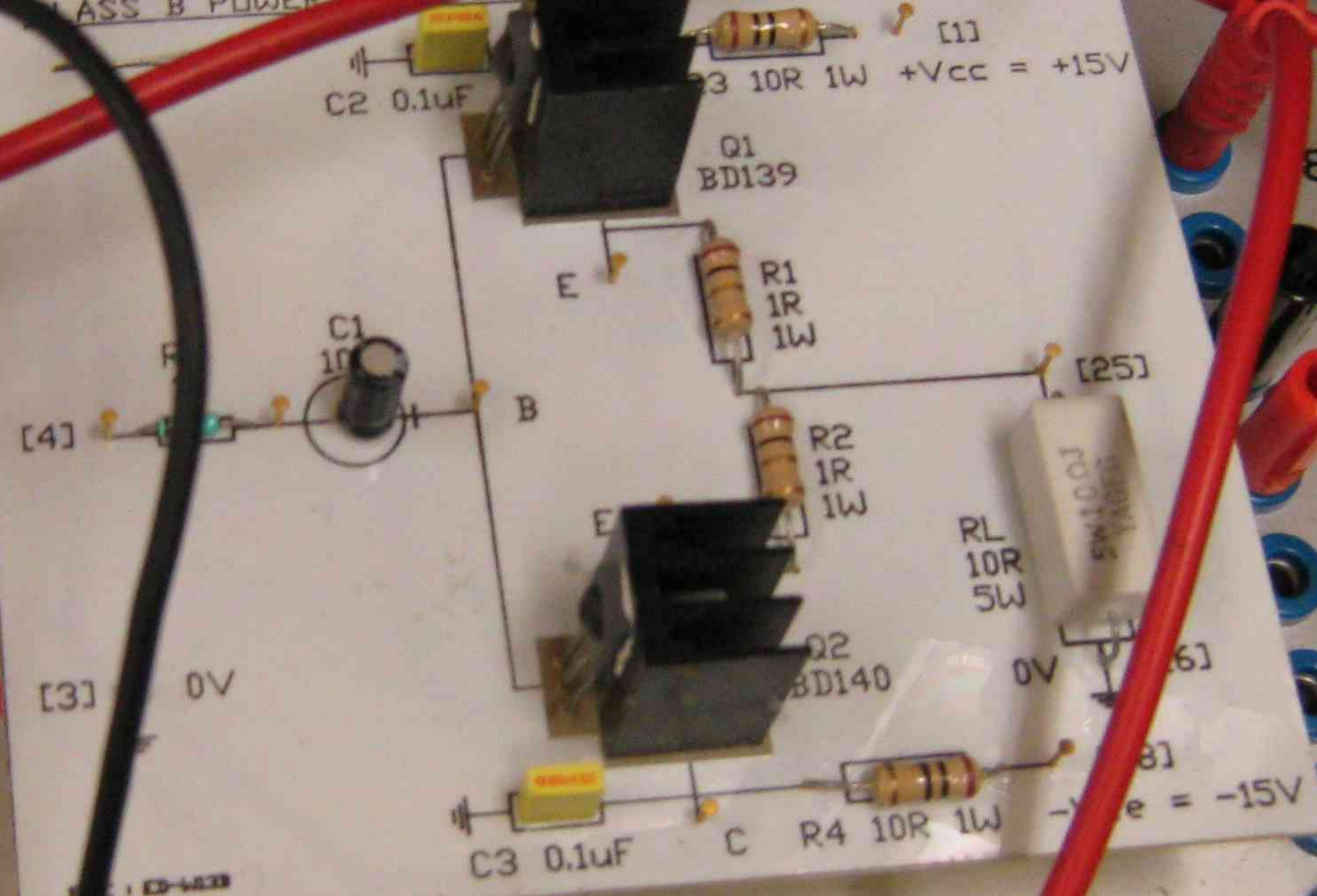
$V_{in} = 15V$

$I_{out} = 100mA$

Power supply

RE CABLE HERE

SIT ULTIMO INDUSTRIAL ELECTRONICS
CLASS B POWER AMPLIFIER
ELECTRONIC DEVICES
6016A # WEEK 13B



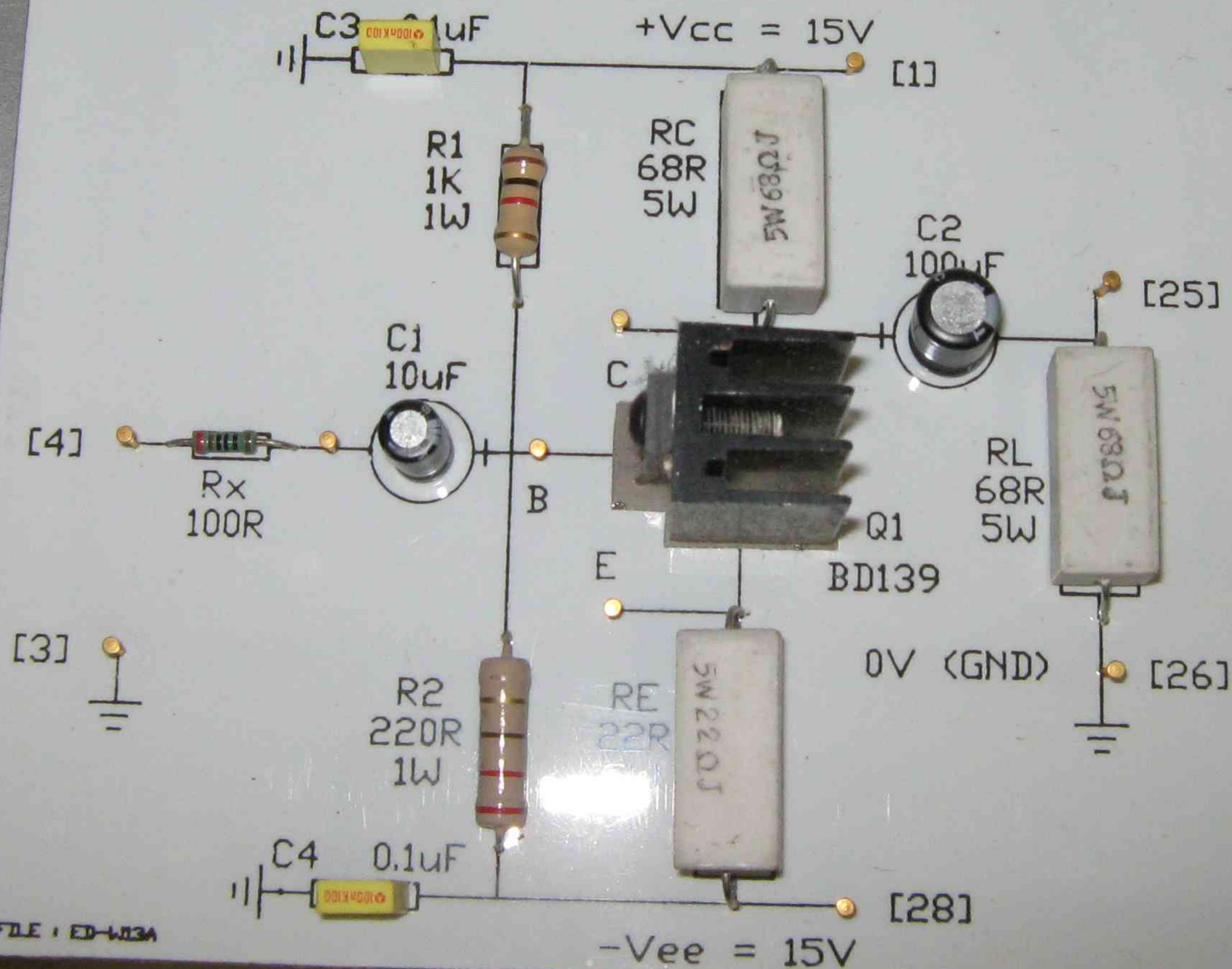
2
3
4
5
6
7
8
9

10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25



SIT ULTIMO INDUSTRIAL ELECTRONICS
CLASS A POWER AMPLIFIER

ELECTRONIC DEVICES
6016A # WEEK 13A



FILE: ED-W13A

SYDNEY INSTITUTE OF TECHNOLOGY, ULTIMO
 ENGINEERING SERVICES TRAINING DIVISION
 ELECTROTECHNOLOGY, INDUSTRIAL ELECTRONICS
 DIGITAL PRINCIPLES 6016B WEEK 14

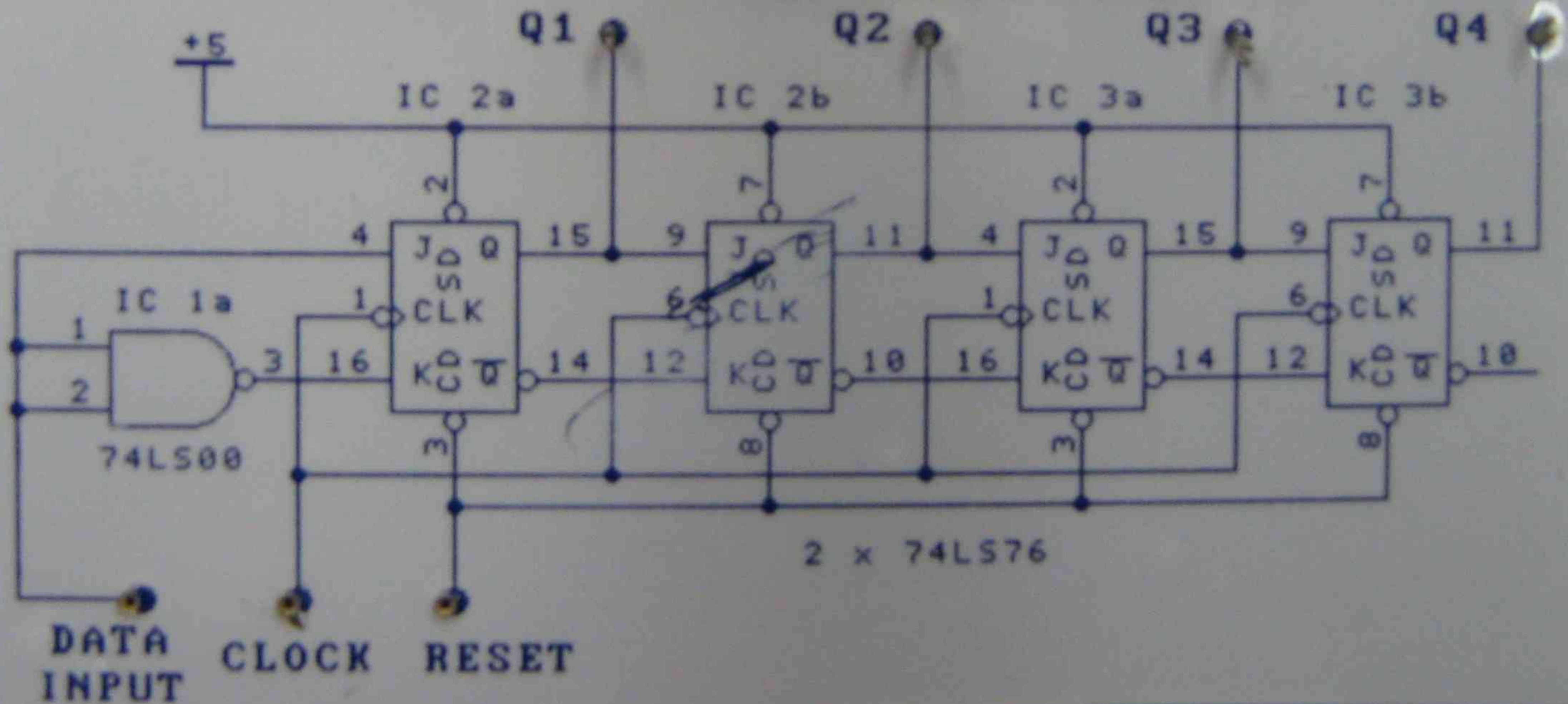
SERIAL TO PARALLEL SHIFT REGISTER

+5V.DC -> PIN 1

COMMON -> PIN 2



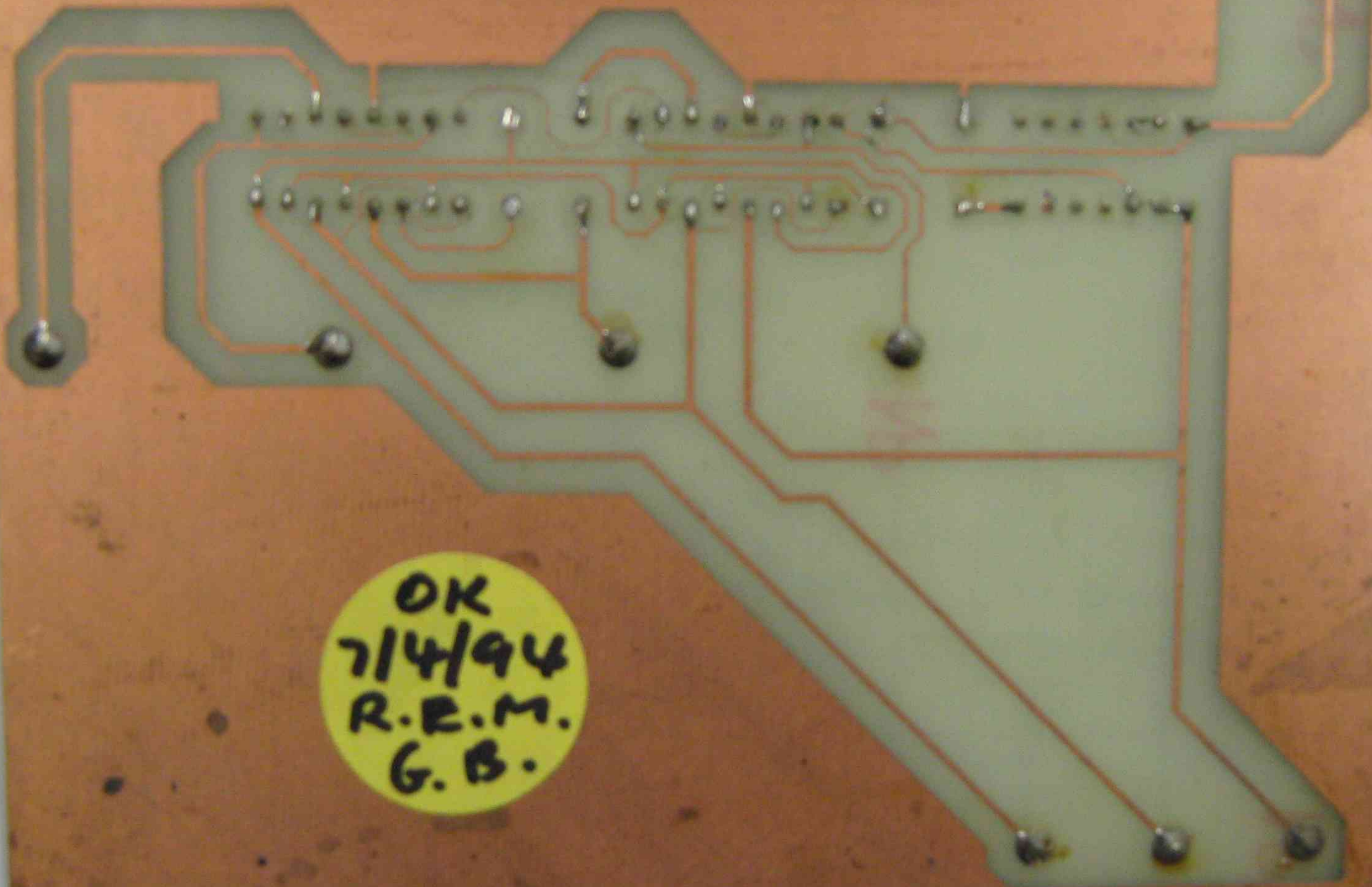
LED INDICATORS



6016B DIGITAL PRINCIPLES
WEEK 14

GND/1M0

VCC +5V

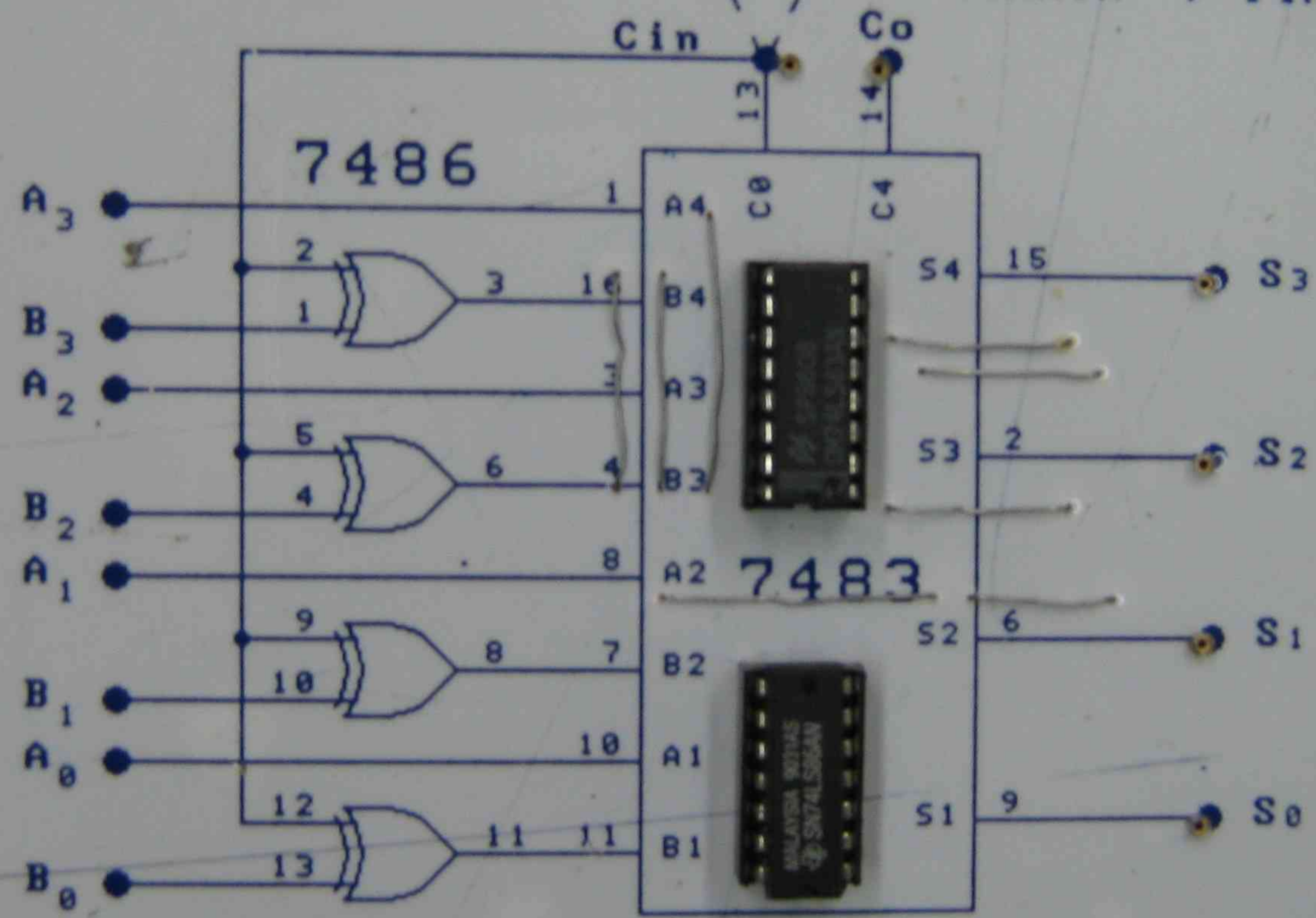


OK
7/4/94
R.E.M.
G.B.

SYDNEY INSTITUTE OF TECHNOLOGY, ULTIMO
 ENGINEERING SERVICES TRAINING DIVISION
 ELECTROTECHNOLOGY, INDUSTRIAL ELECTRONICS
 DIGITAL PRINCIPLES 6016B WEEK 5 PART 2

ADDER/2's COMPLEMENT SUBTRACTOR

+5V. DC -> PIN 1 SUBTRACT ADD COMMON -> PIN 28



A₃ A₂ A₁ A₀ B₃ B₂ B₁ B₀

FILE REF: 6016B5-2

OK
8-4-94
[Signature]

B₀

B₁

B₂

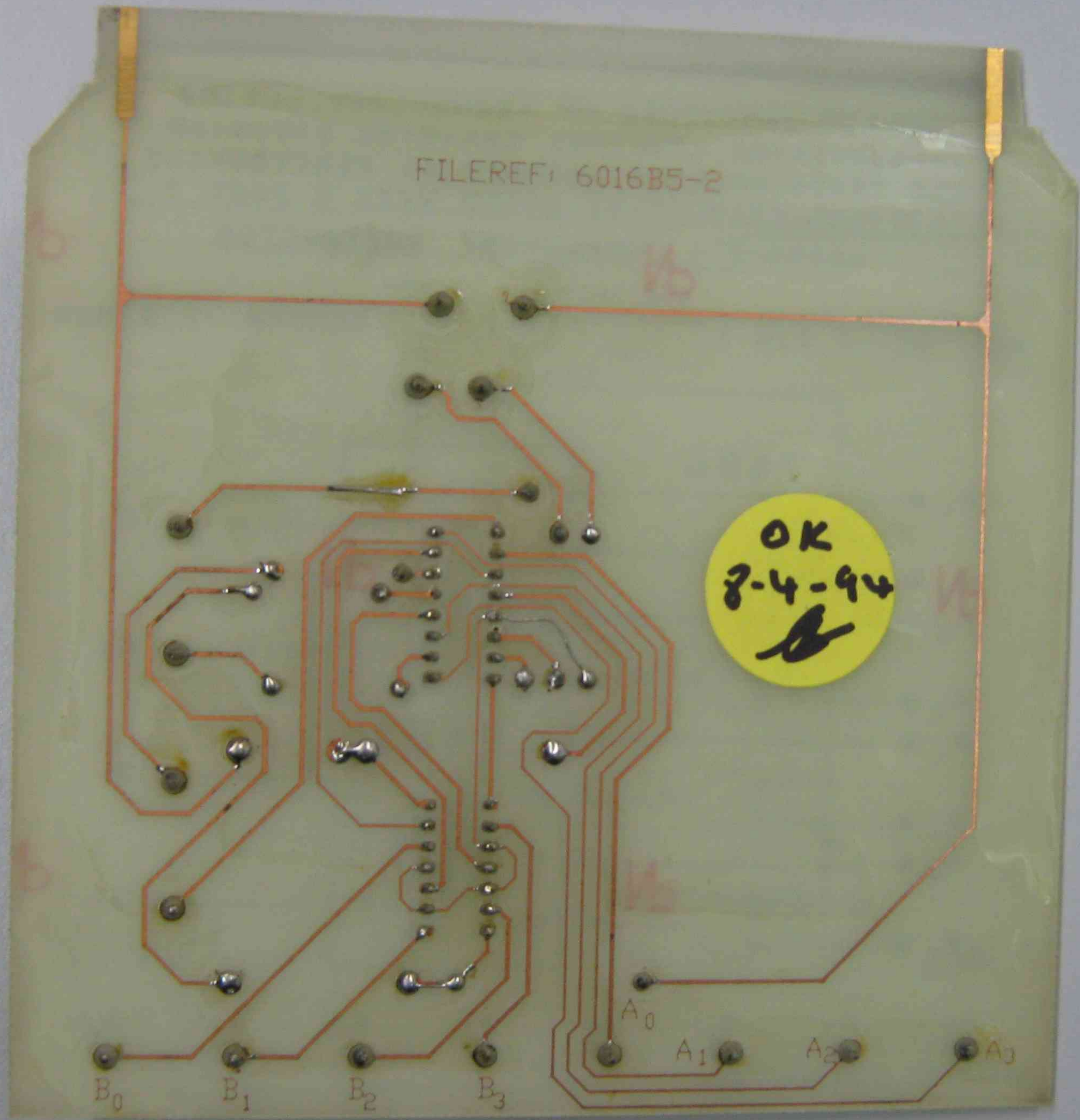
B₃

A₀

A₁

A₂

A₃

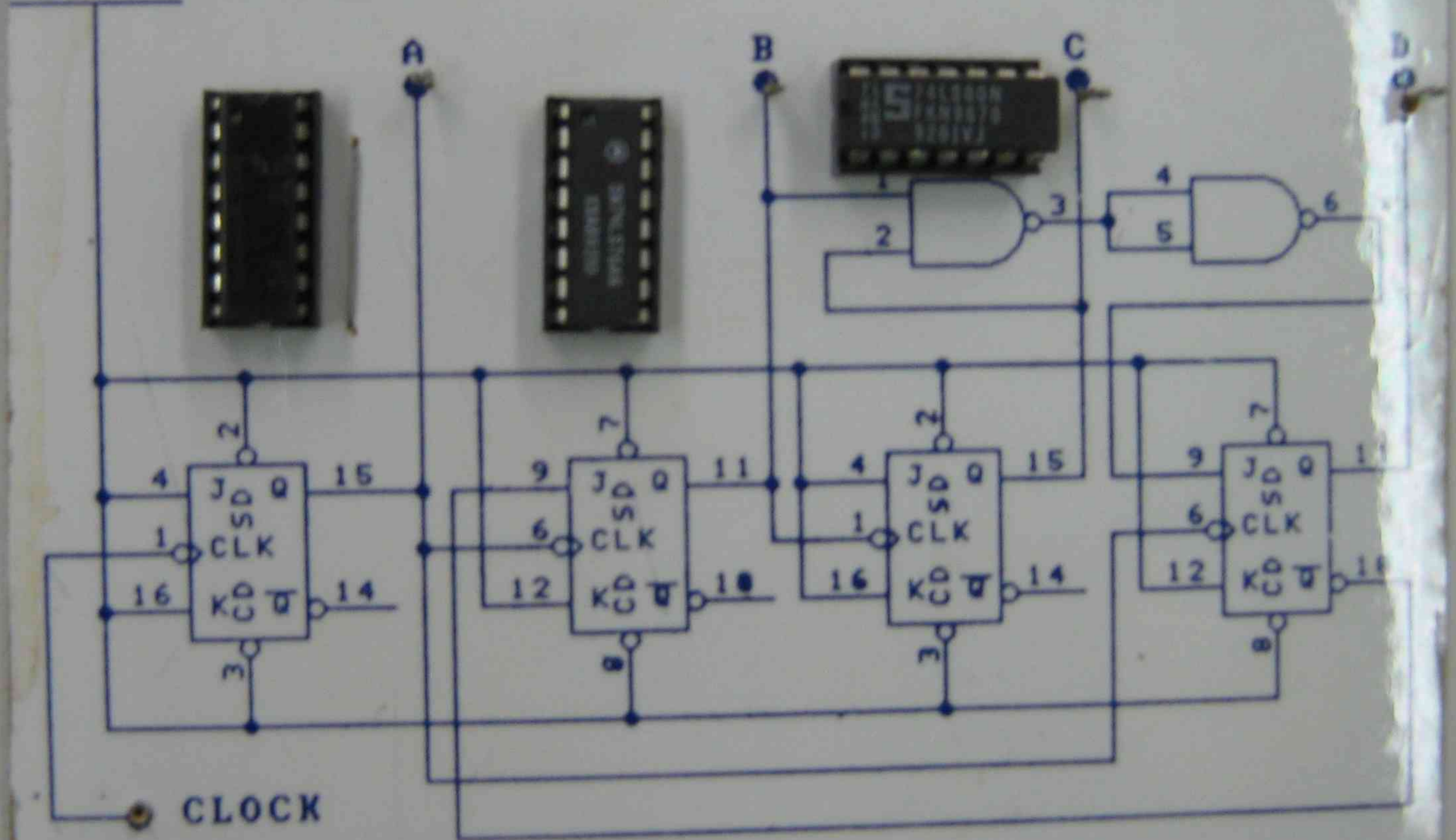


SYDNEY INSTITUTE OF TECHNOLOGY, ULTIMO
ENGINEERING SERVICES TRAINING DIVISION
ELECTROTECHNOLOGY, INDUSTRIAL ELECTRONICS
DIGITAL PRINCIPLES 6016B WEEK 15 PART 2

BCD (Modulo 10) RIPPLE COUNTER

+5V.DC -> PIN 1

COMMON -> PIN 28

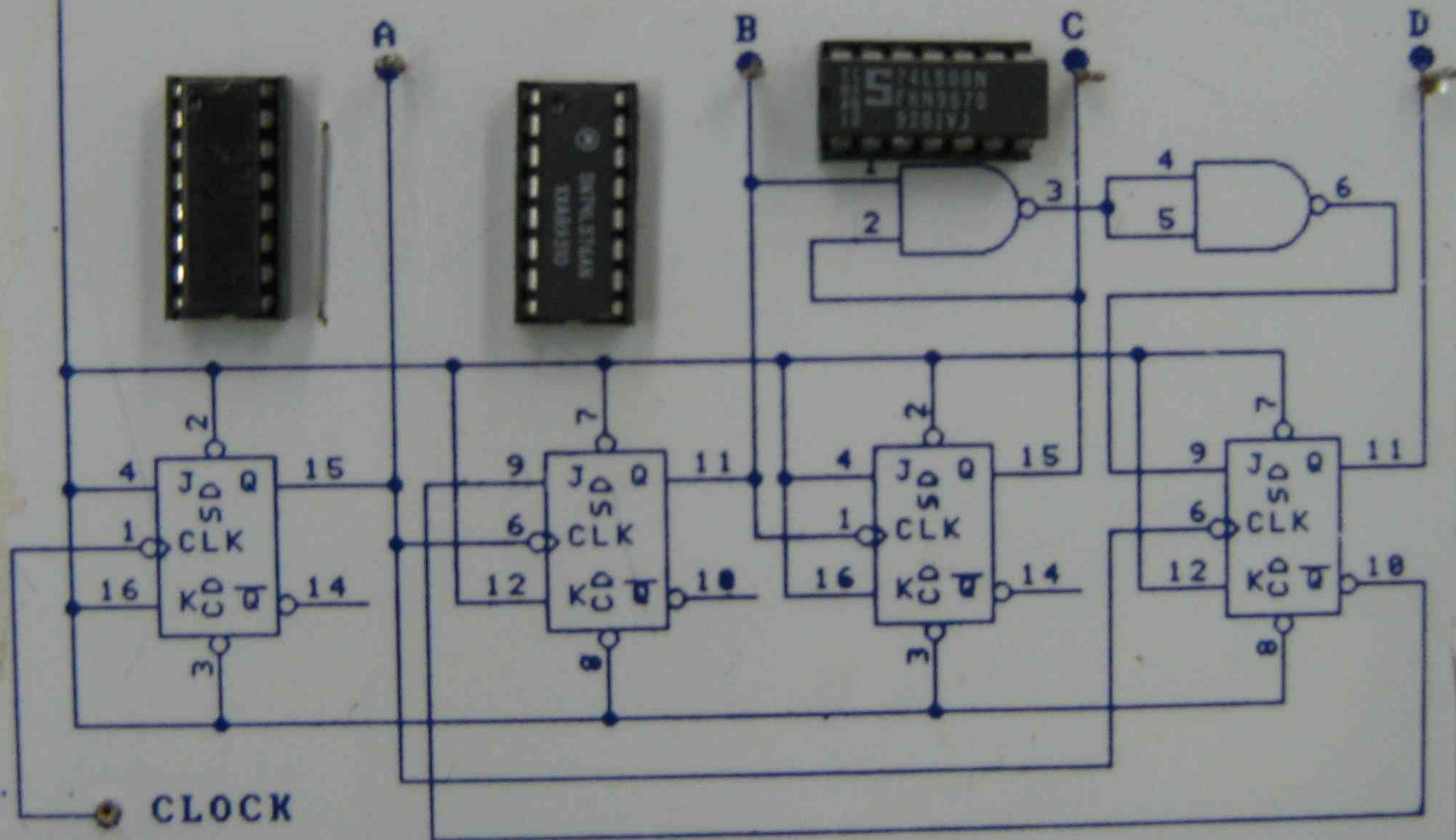


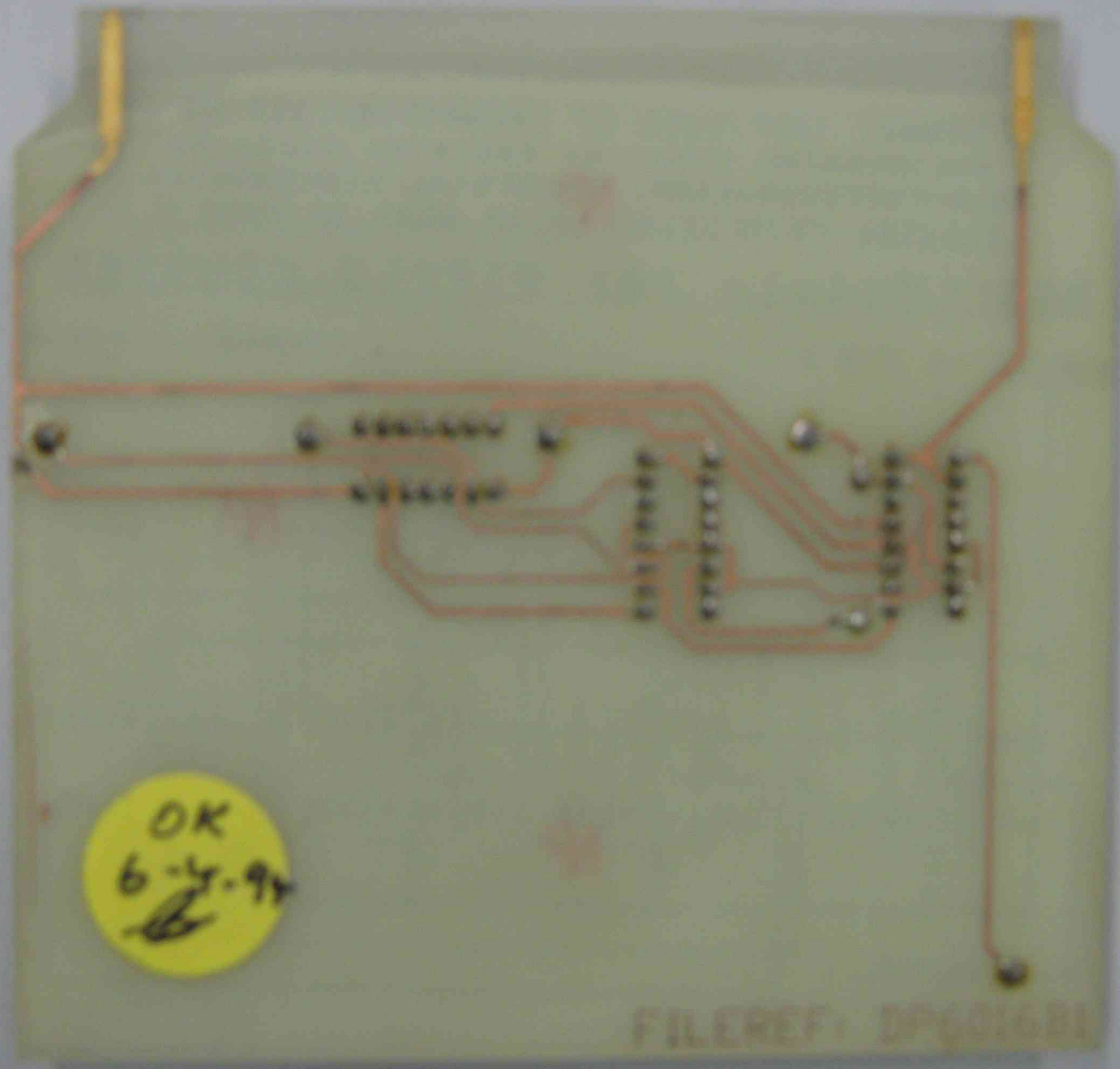
SYDNEY INSTITUTE OF TECHNOLOGY, ULTIMO
ENGINEERING SERVICES TRAINING DIVISION
ELECTROTECHNOLOGY, INDUSTRIAL ELECTRONICS
DIGITAL PRINCIPLES 6016B WEEK 15 PART 2

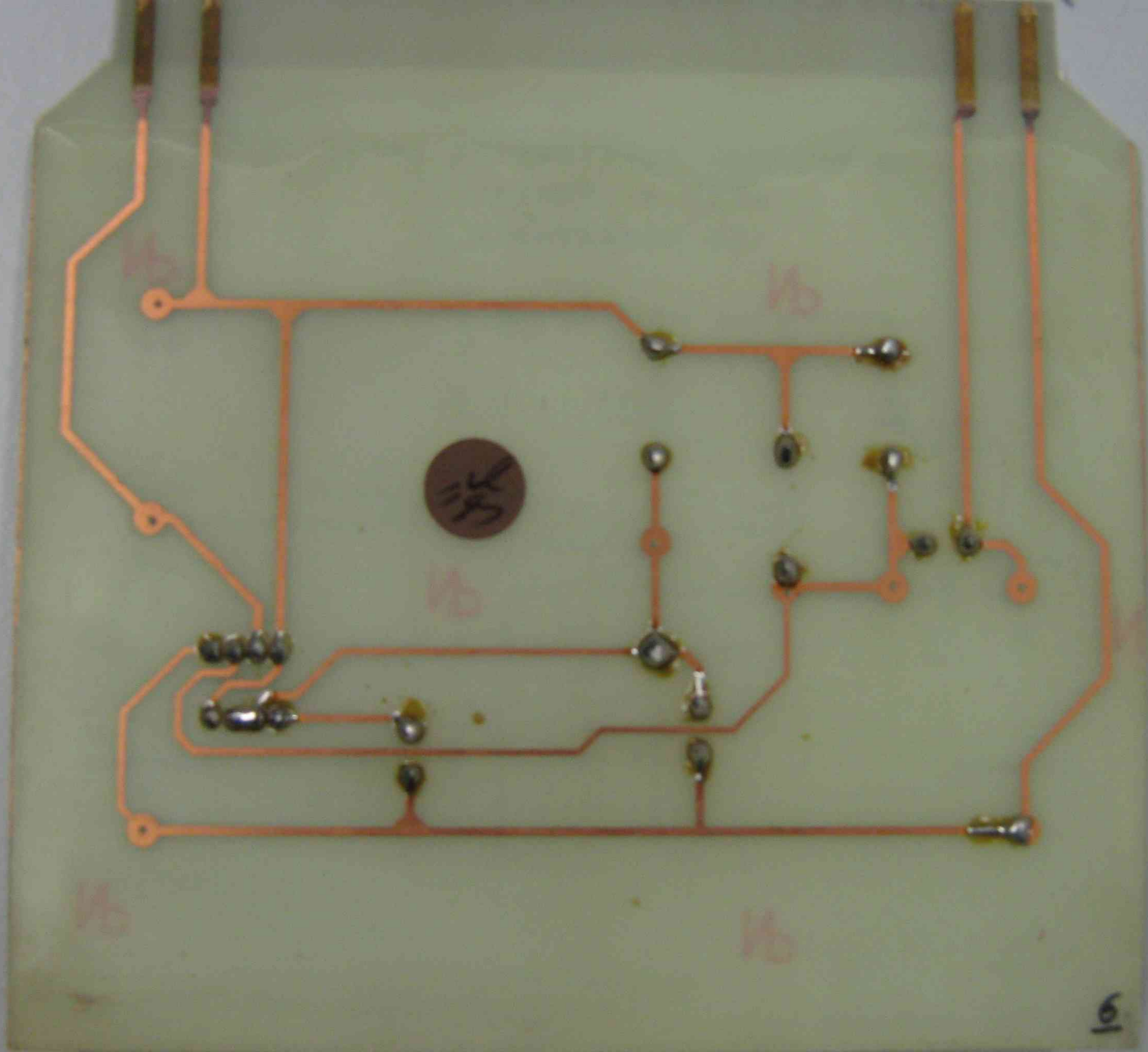
BCD (Modulo 10) RIPPLE COUNTER

+5V.DC -> PIN 1

COMMON -> PIN 28



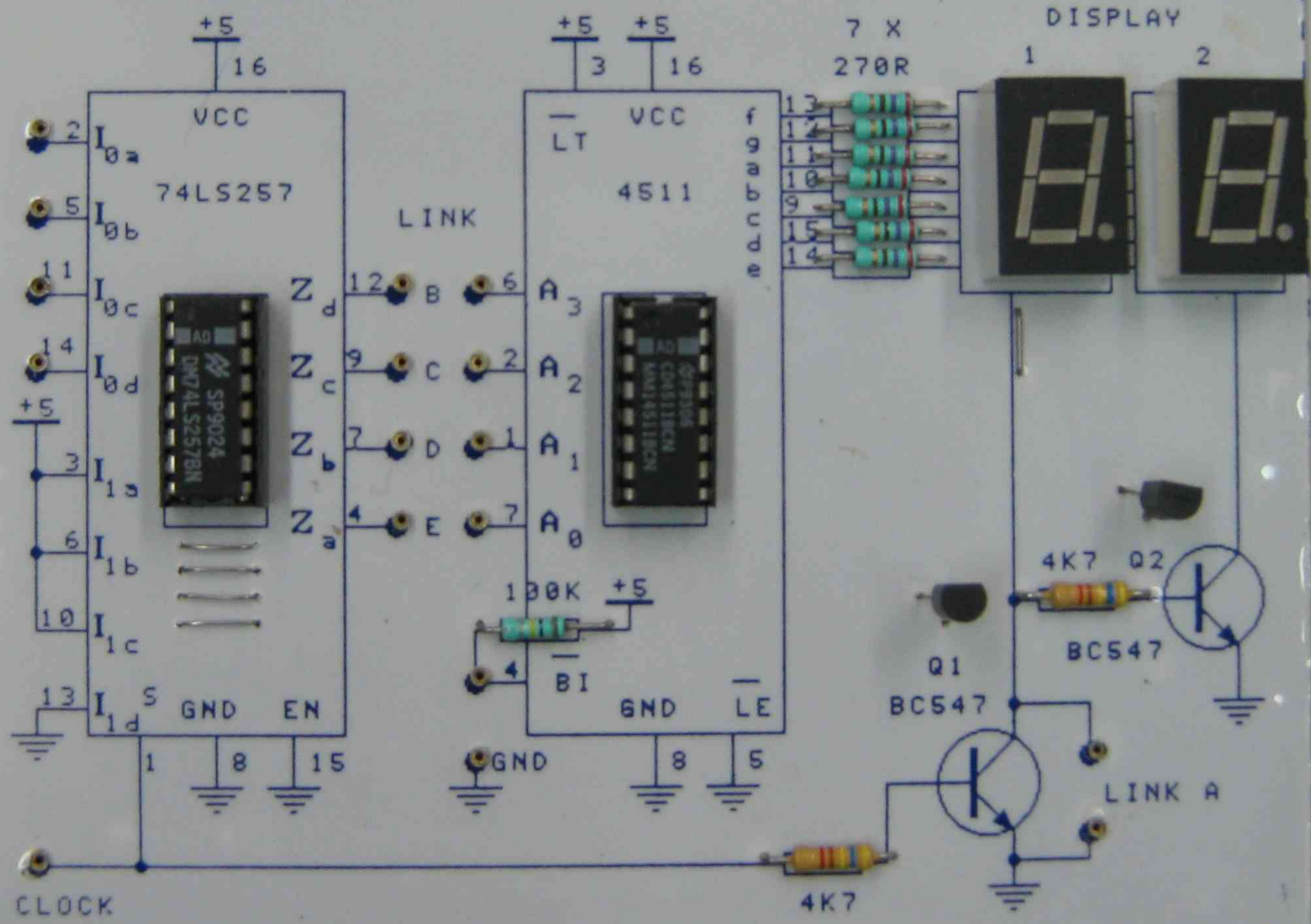




SYDNEY INSTITUTE OF TECHNOLOGY, ULTIMO
 ENGINEERING SERVICES TRAINING DIVISION
 ELECTROTECHNOLOGY, INDUSTRIAL ELECTRONICS
 DIGITAL PRINCIPLES 6016B WEEK 16

DISPLAYS AND DRIVERS

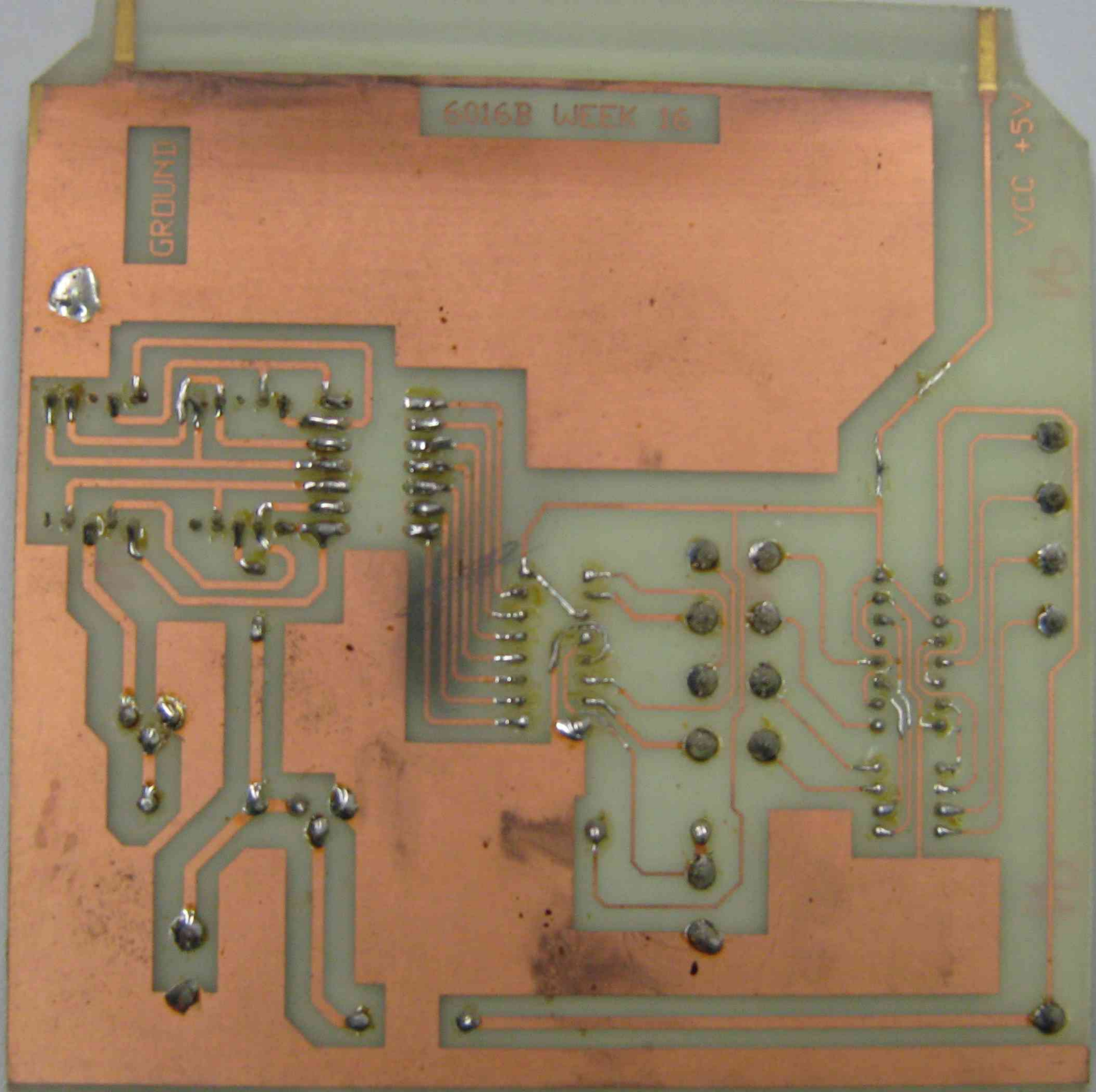
1 - VCC +5V
 28 - GROUND



6016B WEEK 16

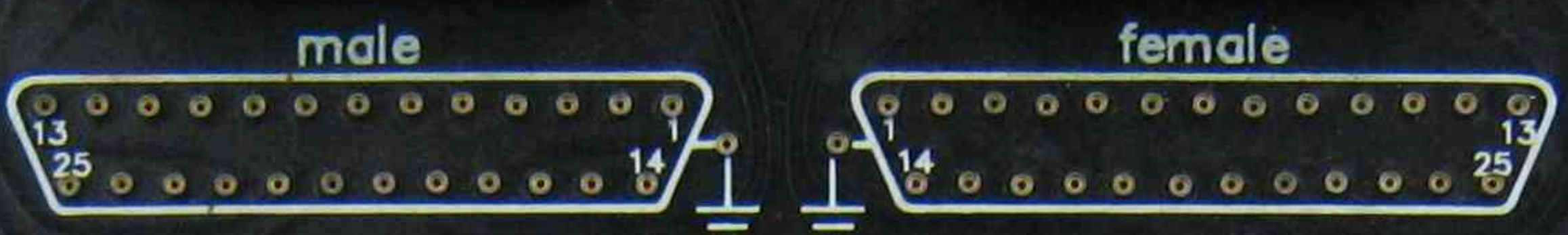
GROUND

VCC +5V





POWER



UC-02 D Sub CONNECTOR



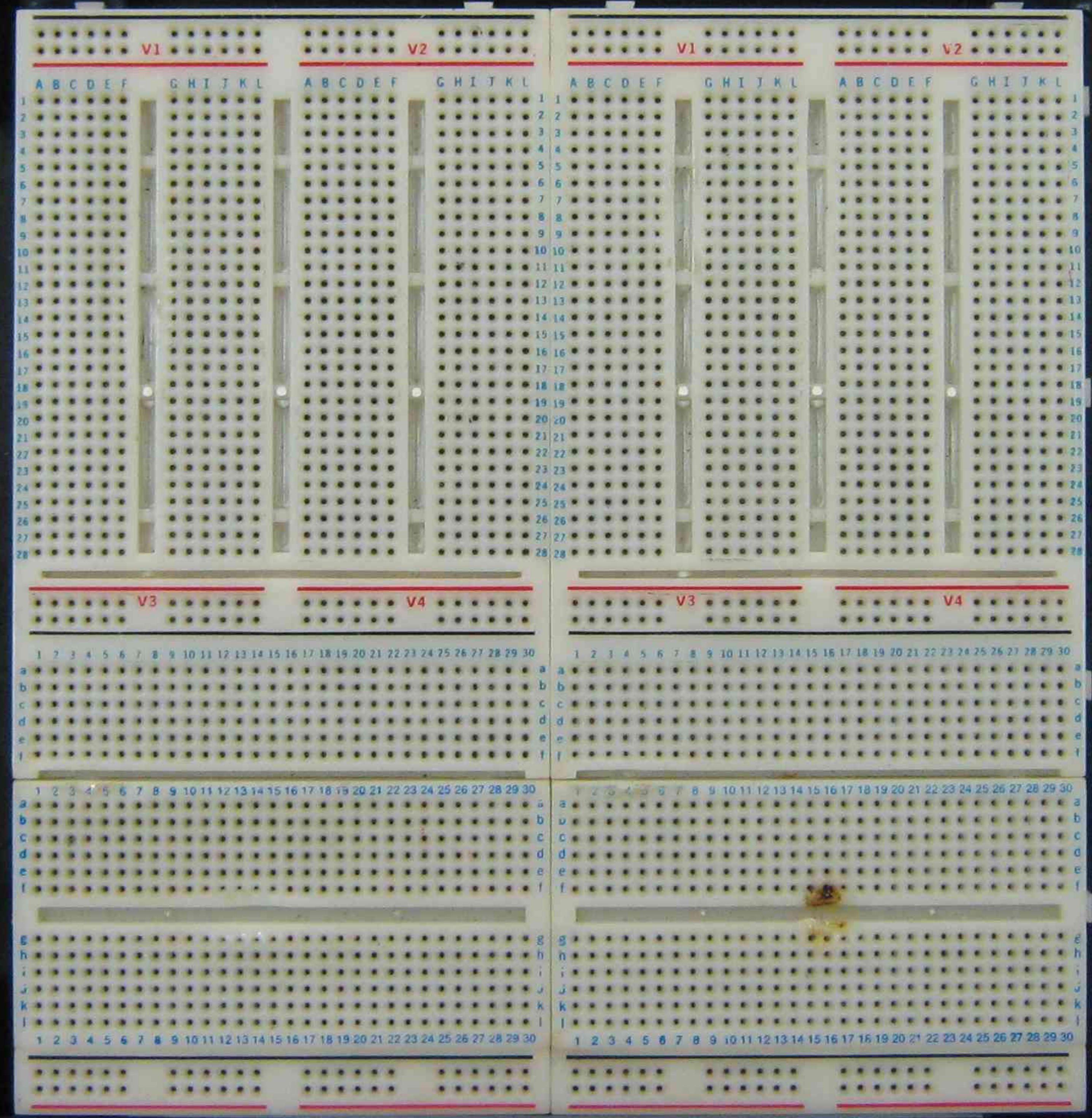
HI : RED
 LO : GREEN
 OPEN: NO DISPLAY

4 0
 5 1
 6 2
 7 3

8 BITS LED DISPLAYS



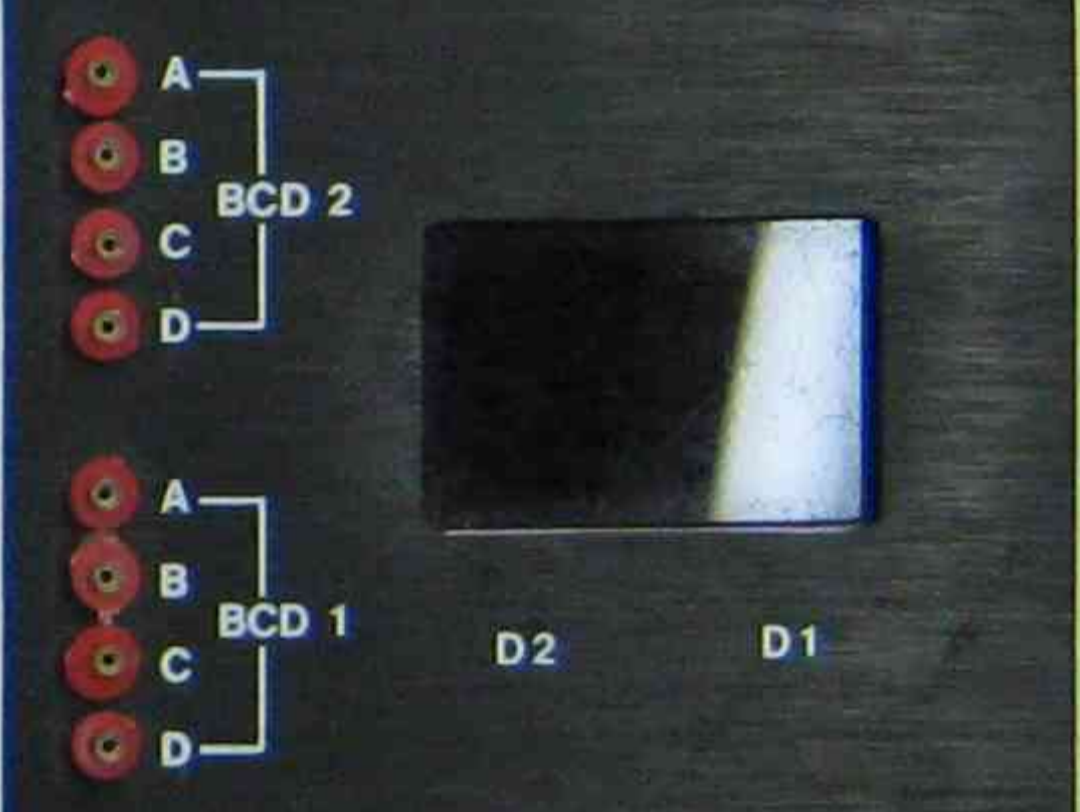
DC POWER



MODE SELECTOR



PULSE GENERATOR



DIGITAL DISPLAYS



PULSER SWITCHES



8 BITS DATA SWITCHES



DIGITAL PROBE

240V G.P.O.

240V G.P.O.

V MAIN
WITCH
P2

D E

DIGITAL COUNTER TYPE TSA 6634/2

H. 1. 16

00000

SENSITIVITY
75mV - 250V

FREQUENCY [KHz] PERIOD CYCLES

Hz [x100] [x10] [x1] [x0.1]

1 10 100

COUNT TIME

POWER OFF

MAX.

RESET ZERO

DISPLAY TIMER

MAX.

INF. AUTO

COUNT/TIME GATE INPUT

+V -V

PERIOD/TIME UNITS

10µS 1mS

SIGNAL INPUT

MADE IN ENGLAND

VENNER ELECTRONICS LIMITED

240V G.P.O.

240V G.P.O.

10V MAIN SWITCH 2

H. 1. 16

DIGITAL COUNTER TYPE TSA 6634/2

0064

SENSITIVITY 75mV - 250V

FREQUENCY PERIOD CYCLES

KHz 1

x100 10

x10 100

Hz x1 COUNT

x0.1 TIME

POWER OFF

MAX.

RESET ZERO

DISPLAY TIMER

MAX.

INF. AUTO

COUNT/TIME GATE INPUT

+V -V

PERIOD/TIME UNITS

10µS 1mS

TEST

SIGNAL INPUT

MADE IN ENGLAND

VENNER ELECTRONICS LIMITED

240V G.P.O.

240 V G.

H. 1. 16

DIGITAL COUNTER TYPE TSA 6634/2

SENSITIVITY
75mV - 250V

1 3 3 0

FREQUENCY
KHz
x100

PERIOD
CYCLES
1
10
100

COUNT
TIME

POWER
OFF

SIGNAL
INPUT

TEST

MADE IN ENGLAND

RESET
ZERO

DISPLAY TIMER

INF. AUTO

VENNER ELECTRONICS LIMITED

DIGITAL COUNTER TYPE TSA 6634/2

H. 1. 16

31604

SENSITIVITY
75mV - 250V

FREQUENCY PERIOD CYCLES

KHz 1

x100 10

x10 100

Hz x1 COUNT

x0.1 TIME

POWER OFF

MAX

RESET ZERO

DISPLAY TIMER

MAX.

INF. AUTO

COUNT/TIME GATE INPUT

+V

-V

PERIOD/TIME UNITS

10µS 1mS

TEST

SIGNAL INPUT

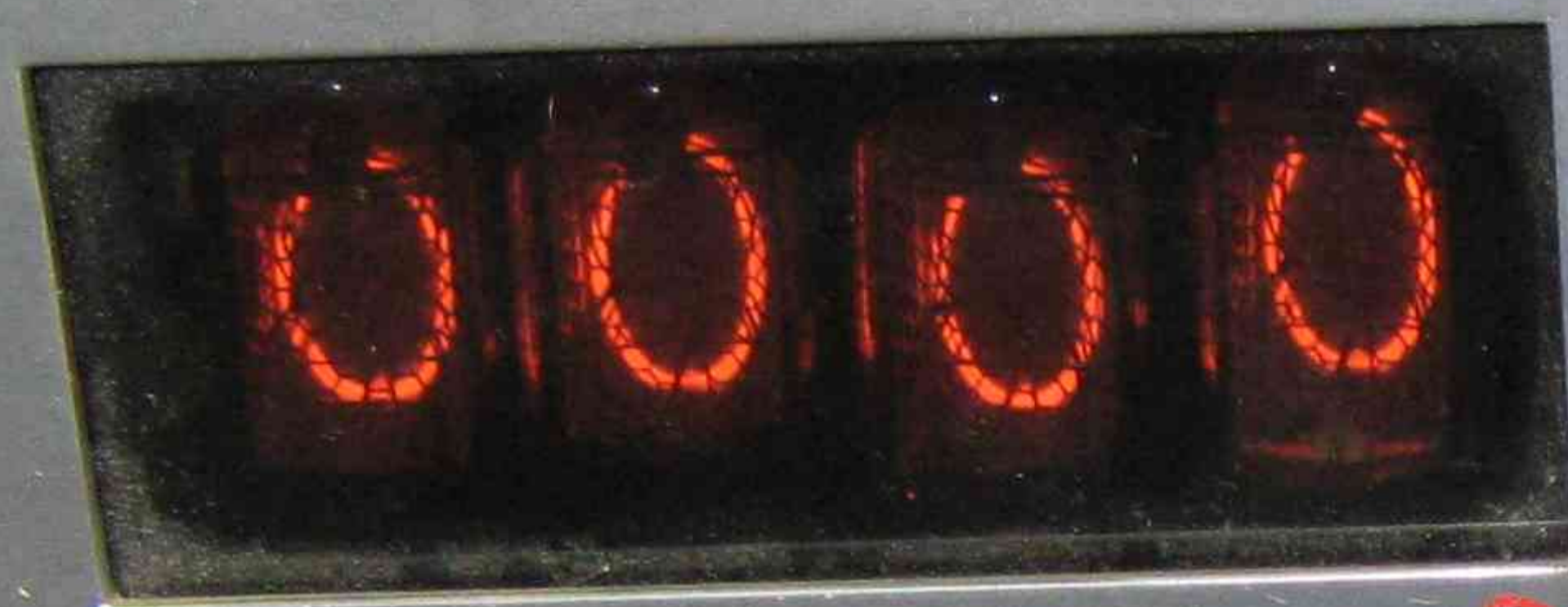
VENNER ELECTRONICS LIMITED

MADE IN ENGLAND

DIGITAL COUNTER TYPE TSA 6634/2

H. 1. 16

SENSITIVITY
75mV - 250V



FREQUENCY PERIOD CYCLES

KHz 1

x100 10

x10 100

Hz COUNT

x1 TIME

x0.1

POWER OFF

MAX.

RESET ZERO

DISPLAY TIMER

MAX.

INF. AUTO

COUNT/TIME GATE INPUT

+V -V

PERIOD/TIME UNITS

10µS 1mS

SIGNAL INPUT

TEST

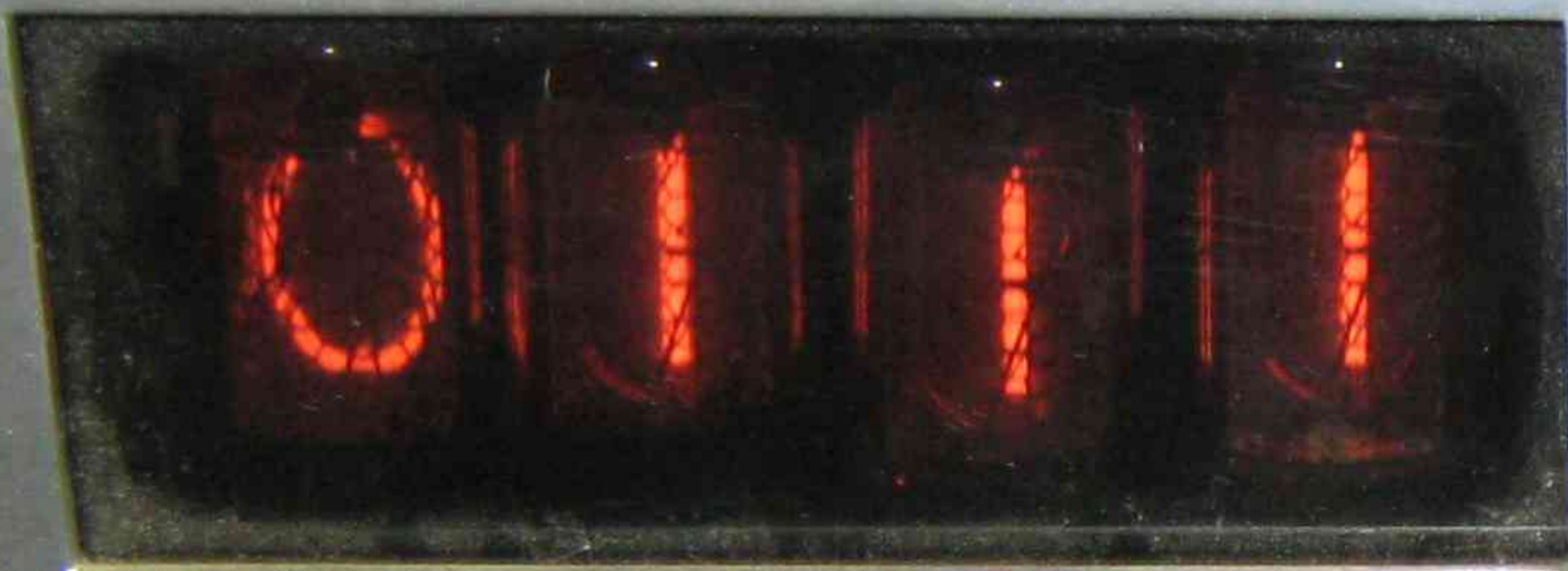
VENNER ELECTRONICS LIMITED

MADE IN ENGLAND

DIGITAL COUNTER TYPE TSA 6634/2

H. 1. 16

SENSITIVITY
75mV - 250V



FREQUENCY PERIOD CYCLES

KHz 1

x100 10

x10 100

Hz x1 COUNT

x0.1 TIME

POWER OFF

MAX.

RESET ZERO

DISPLAY TIMER

MAX.

Handwritten red text: 'i' with a red arrow pointing to the 'COUNT/TIME GATE INPUT' label.

COUNT/TIME GATE INPUT

+V

-V

PERIOD/TIME UNITS

10µS 1mS

TEST

SIGNAL INPUT

VENNER ELECTRONICS LIMITED

MADE IN ENGLAND

DIGITAL COUNTER TYPE TSA 6634/2

H. 1. 16

4 1 9 3 9

SENSITIVITY
75mV - 250V

FREQUENCY PERIOD CYCLES

KHz 1

x100 10

x10 100

Hz x1 COUNT

x0.1 TIME

POWER OFF

MAX.

RESET ZERO

DISPLAY TIMER

MAX.

INF. AUTO

COUNT/TIME GATE INPUT

+V

-V

PERIOD/TIME UNITS

TEST

SIGNAL INPUT

MADE IN ENGLAND

VENNER ELECTRONICS LIMITED

240V MAIN SWITCH No 2

240V G.P.O.

240V G

FORCING UNIT
SERIES 1000
NO. 1000
CENTRE CABLE HERE

band
bt

H. 1. 16

DIGITAL COUNTER TYPE TSA 6634/2

SENSITIVITY
75mV - 250V

70000

FREQUENCY PERIOD CYCLES
x100 x10
Hz
COUNT

POWER OFF
MAX.

RESET ZERO

DISPLAY TIMER
MAX.

INF AUTO

SIGNAL INPUT

TEST

MADE IN ENGLAND

VENNER ELECTRONICS LIMITED

MAIN SWITCH
No 2

240V G.P.O.

240V

H. 1. 16

DIGITAL COUNTER TYPE TSA 6634/2

SENSITIVITY
75mV - 250V

9 3 8

FREQUENCY PERIOD CYCLES
KHz 1 10 100
COUNT

POWER OFF

SIGNAL INPUT

RESET ZERO

DISPLAY TIMER

INF.

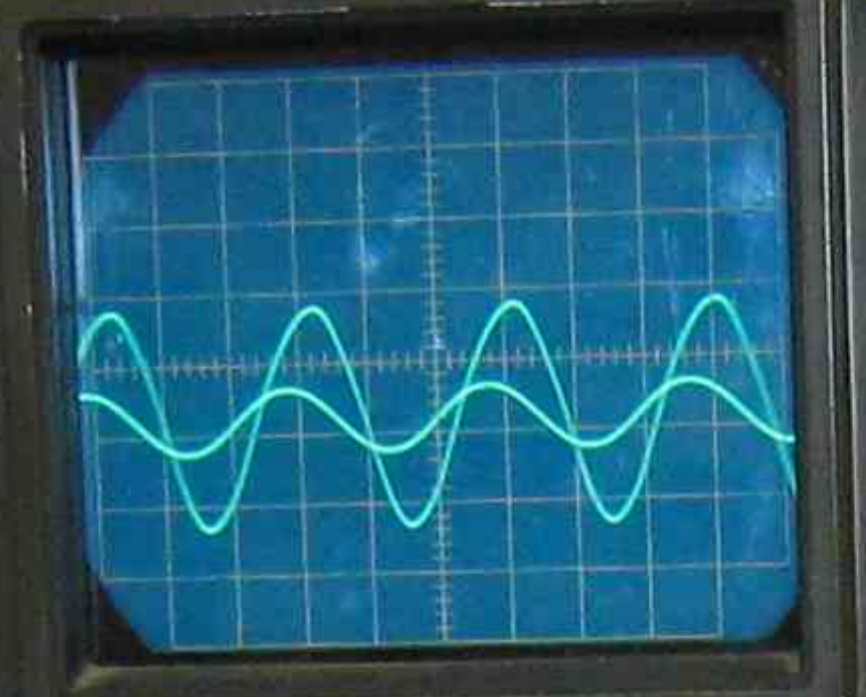
AUTO

MADE IN ENGLAND

VENNER ELECTRONICS LIMITED

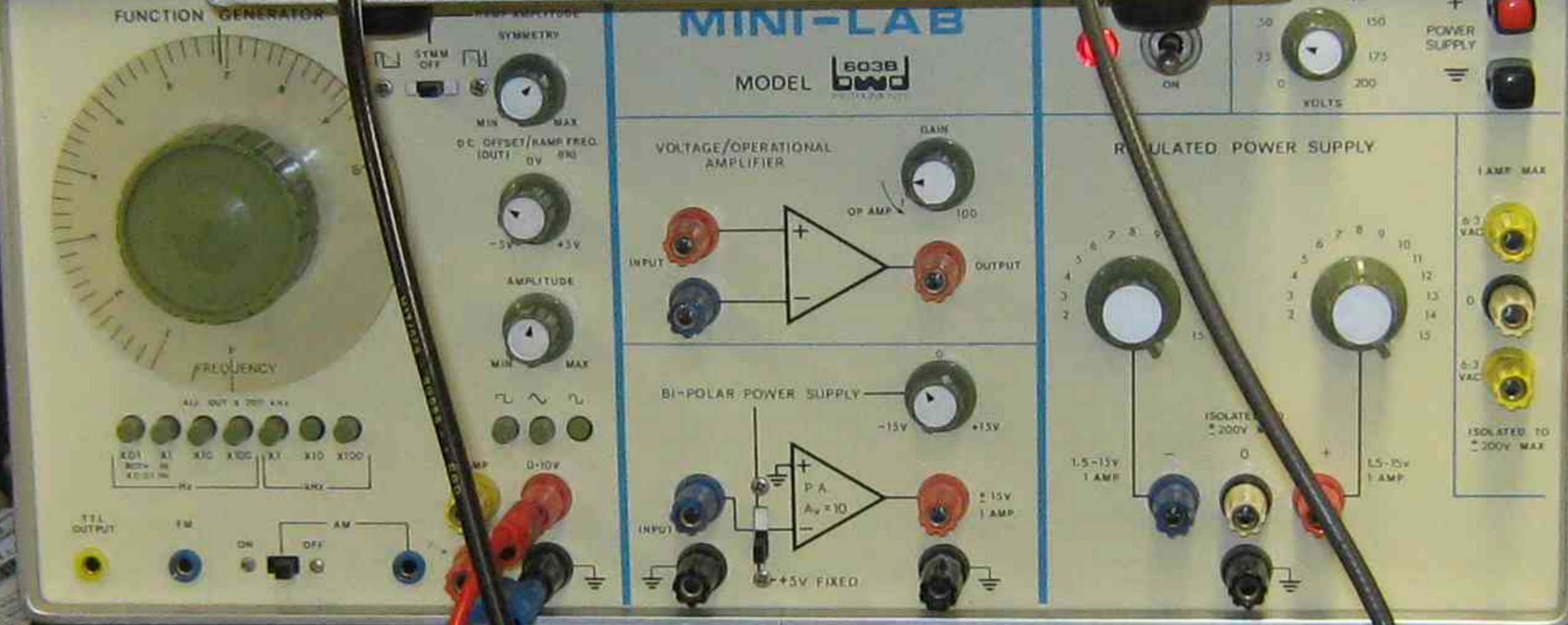
Put equipment away
Hang leads neat and tidy

TRIO 15MHz OSCILLOSCOPE CS-1560A II



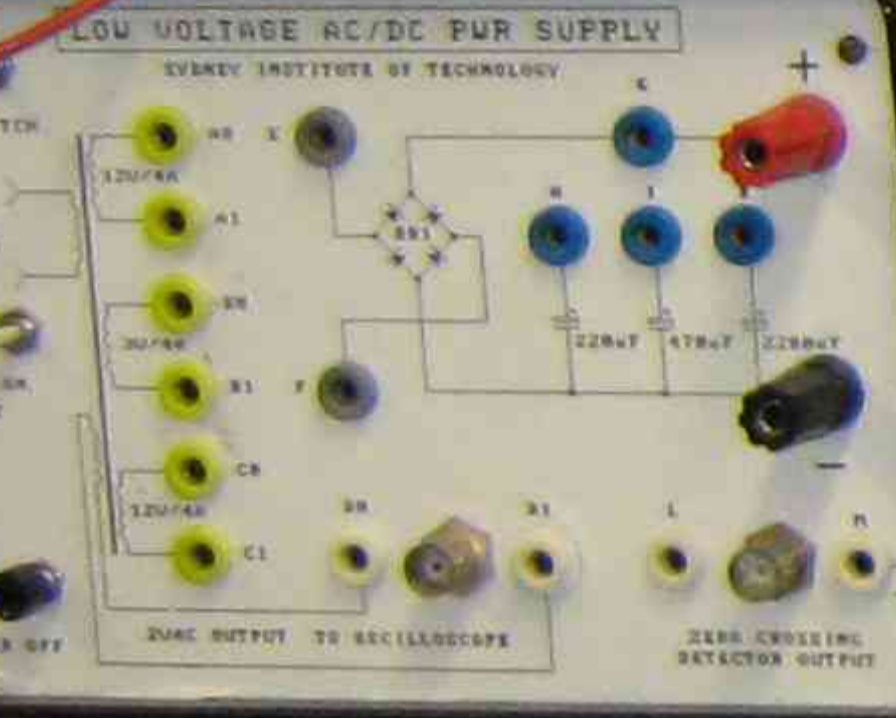
POWER SCALE ILLUM
VARIABLE SWEEP TIME/DIV
TRIGGERING LEVEL
TRIGGERING SOURCE
CH1 or Y VOLTS/DIV
MODE
CH2 or X VOLTS/DIV

MINI-LAB MODEL 603B



FUNCTION GENERATOR
VOLTAGE/OPERATIONAL AMPLIFIER
BI-POLAR POWER SUPPLY
REGULATED POWER SUPPLY

LOW VOLTAGE AC/DC PUR SUPPLY

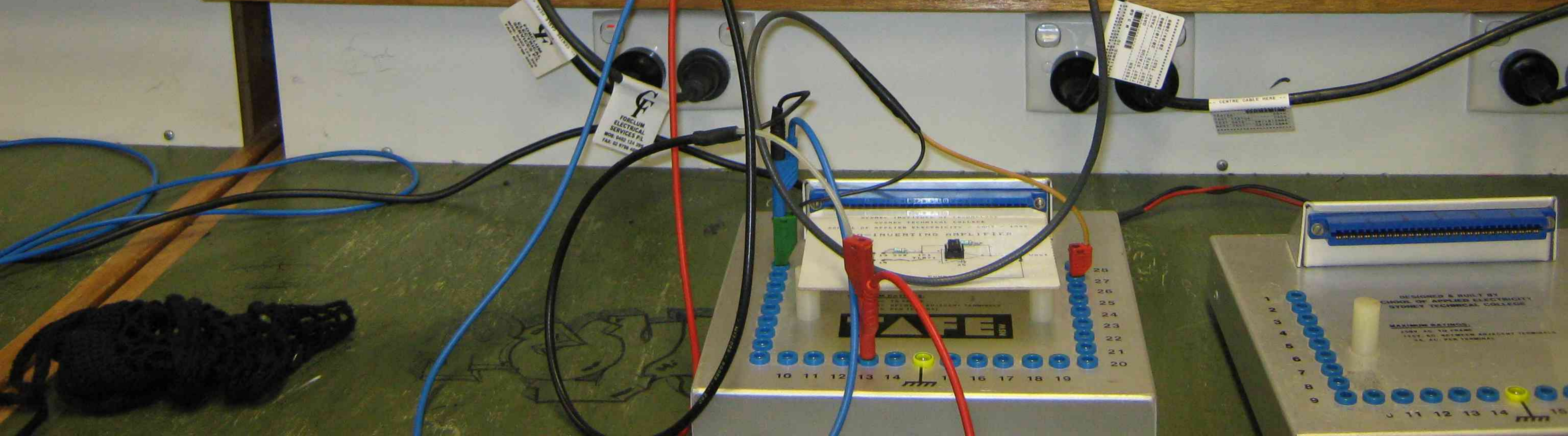


EVERETT INSTITUTE OF TECHNOLOGY

FUNCTION GENERATOR MODEL GFD-8020G



0.000

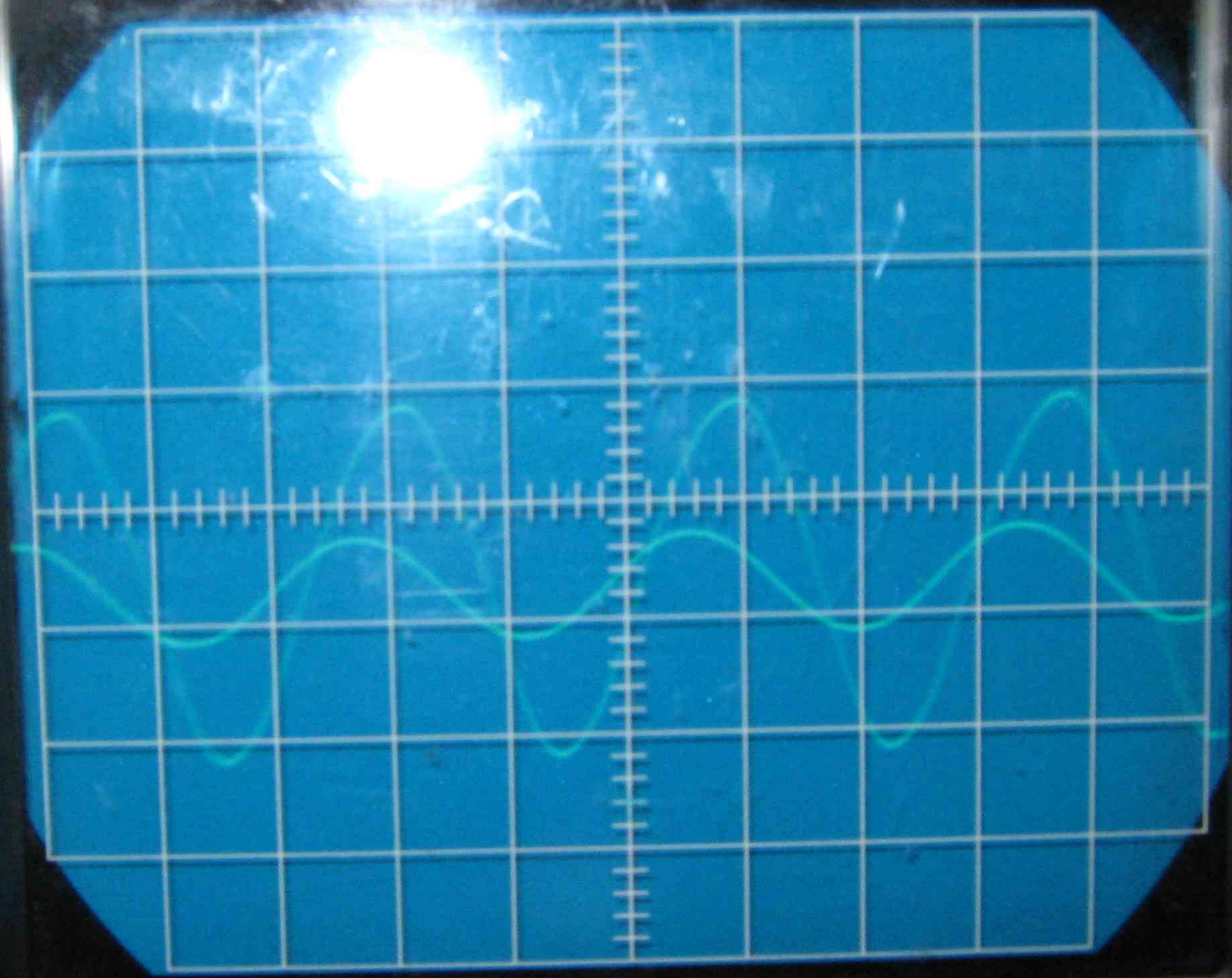


FORCLUM ELECTRICAL SERVICES PTY LTD
110/112 STURGEON RD
SYDNEY NSW 1570

DESIGNED & BUILT BY
CHADLER OF APPLIED ELECTRONICS
SYDNEY TECHNICAL COLLEGE

MAXIMUM RATINGS
DO NOT EXCEED THESE
LIMITS OR NETWORK ELEMENTS
WILL BE DESTROYED

TRIO 15MHz OSCILLOSCOPE CS-150A II

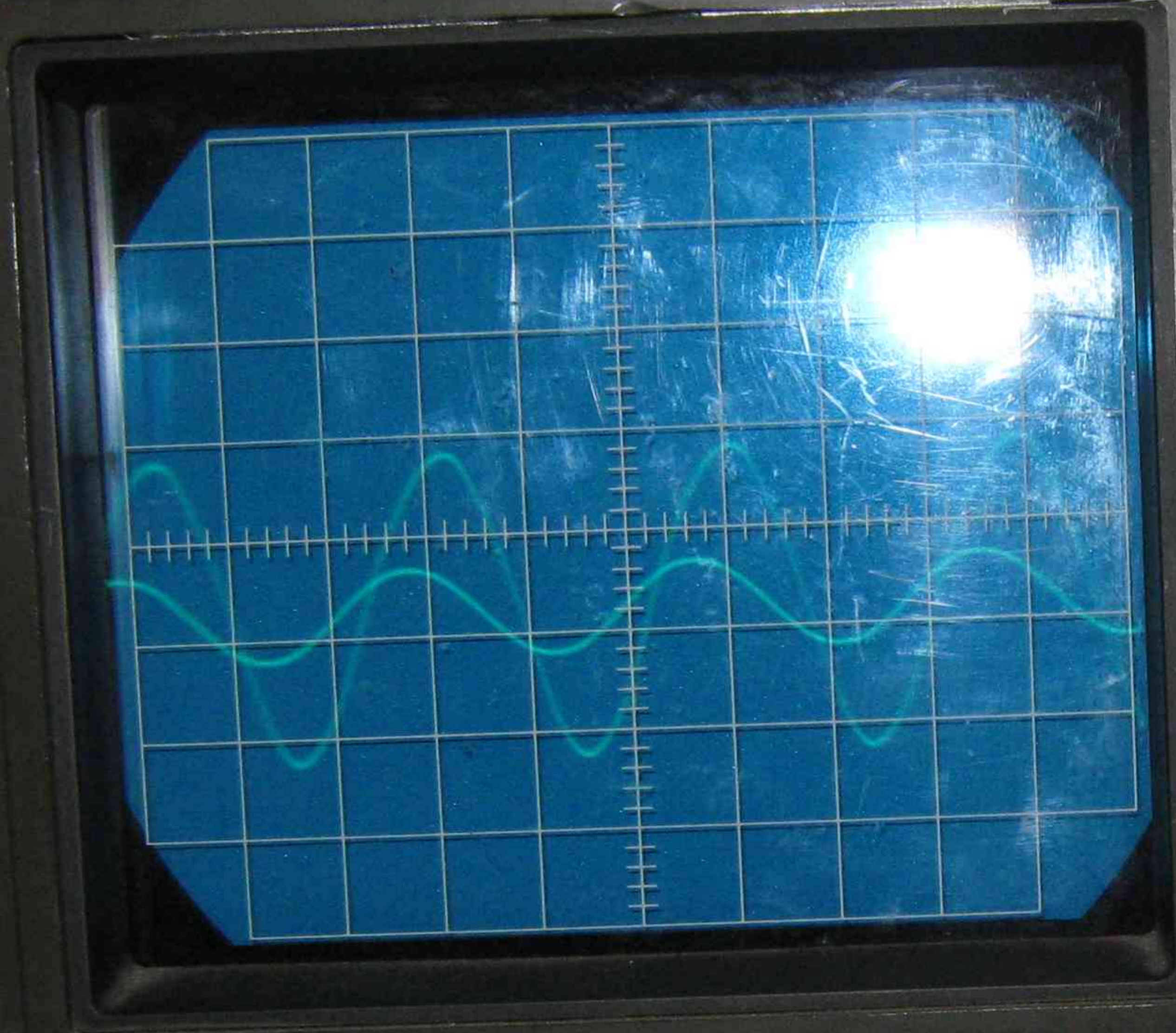


CH1 or Y

VARIABLE VOLTS/DIV

MODE

TRIO 15MHz OSCILLOSCOPE CS-1560A II



POWER SCALE ILLUM



VARIABLE SWEEP TIME/DIV



CAL



POSIT PULL X5 MAG



TRACE ROTATION INTENSITY



TRIGGERING LEVEL PULL AUTO



ASTIG



FOCUS



SYNC NORM TV



SOURCE CH1 CH2 EXT



EXT. TRIG



CH1 or Y

POSITION



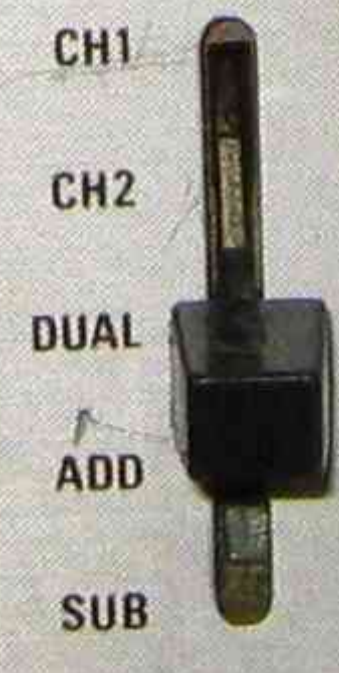
INPUT 1MΩ ≈ 22pF



VARIABLE VOLTS/DIV



MODE



VARIABLE VOLTS/DIV



INPUT 1MΩ ≈ 22pF

POSITION X-Y



DC OFFSET/RAMP FREQ. (OUT) 0V (IN)

VOLTAGE/OPERATIONAL AMPLIFIER

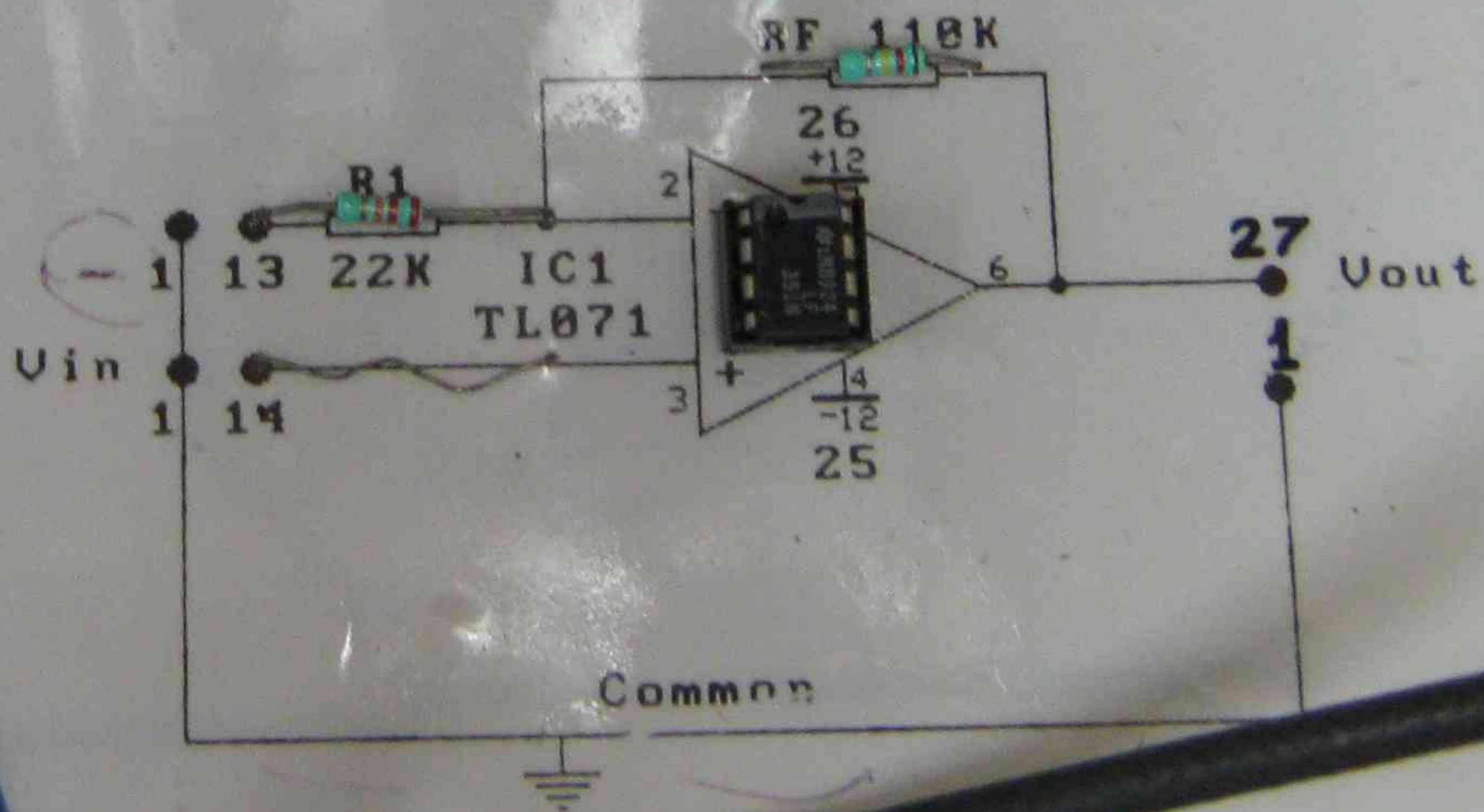
OP AMP

REGULATED POWER SUPPLY

1 AMP

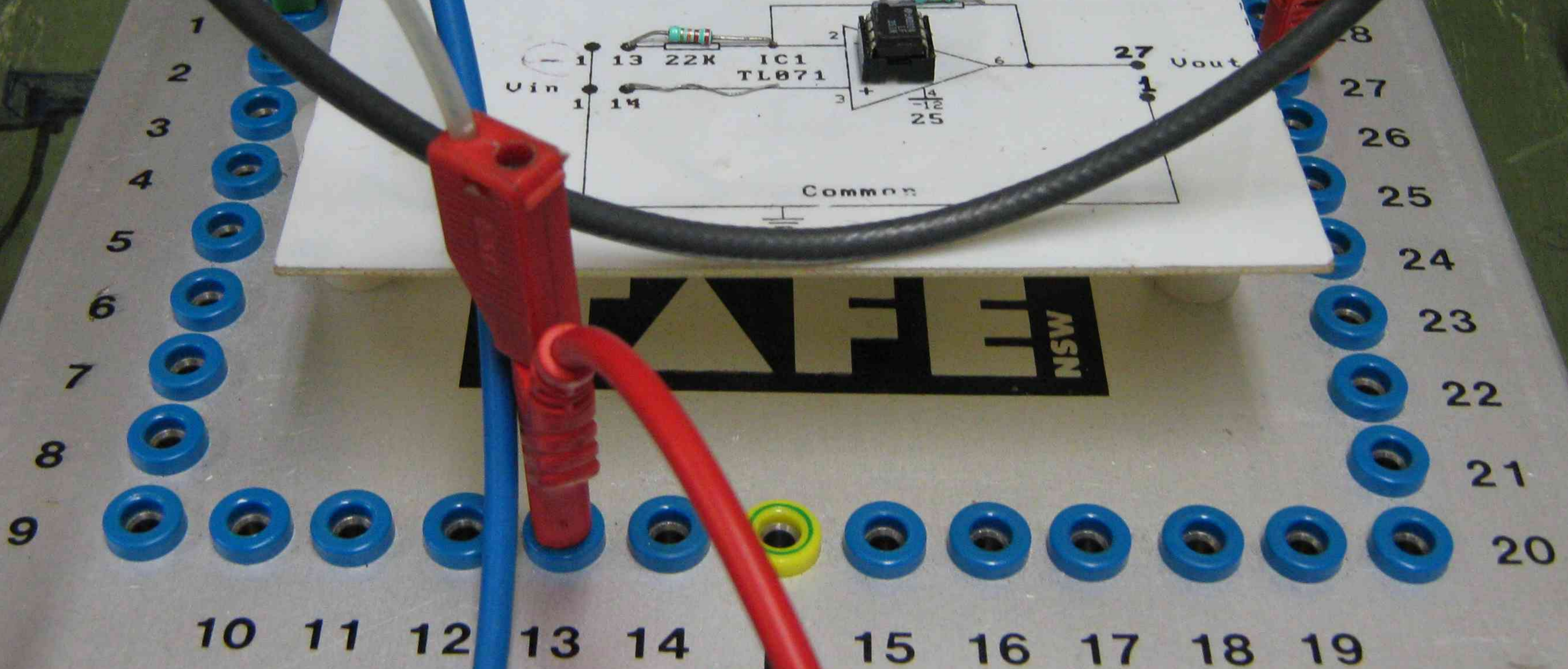
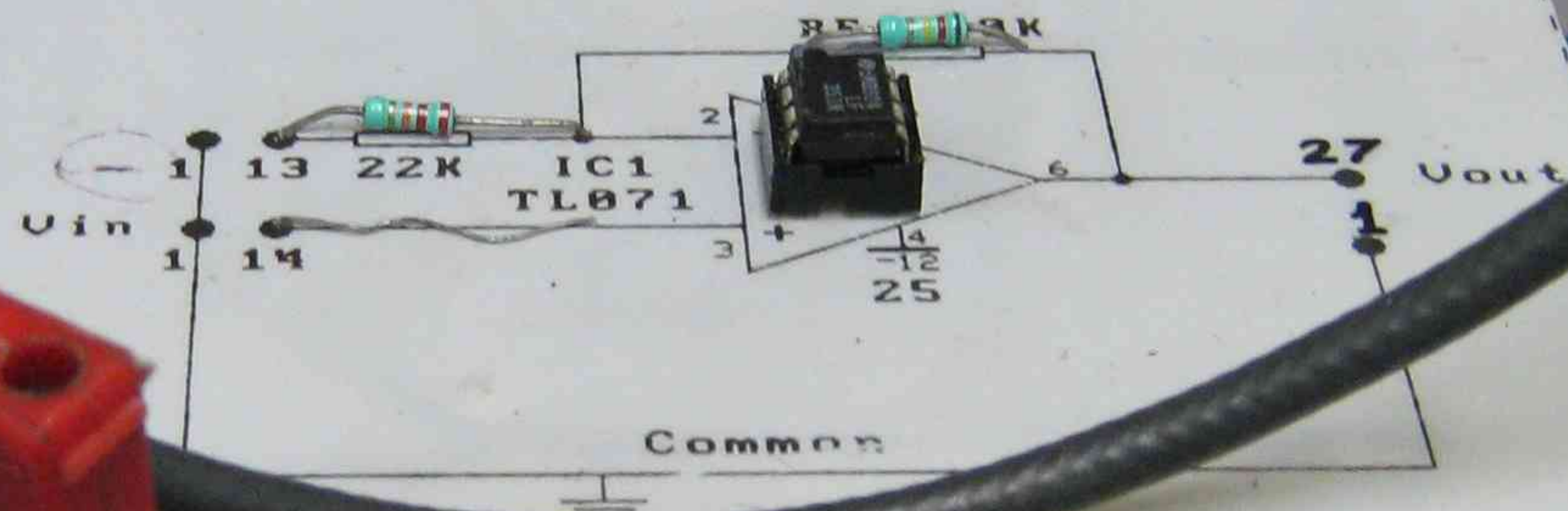
SYDNEY INSTITUTE OF TECHNOLOGY
SYDNEY TECHNICAL COLLEGE
SCHOOL OF APPLIED ELECTRICITY / CAMS / 1992

NON-INVERTING AMPLIFIER



SYDNEY INSTITUTE OF TECHNOLOGY
SYDNEY TECHNICAL COLLEGE
SCHOOL OF APPLIED ELECTRICITY / CAMS / 1992

NON-INVERTING AMPLIFIER



CH1 or Y

POSITION

INPUT 1MΩ = 22pF

DC BAL

VARIABLE VOLTS/DIV

MODE

CH1

CH2

DUAL

ADD

SUB

VARIABLE VOLTS/DIV

CH2 or X

POSITION X-Y

INPUT 1MΩ = 22pF

DC BAL

AC

GND

DC

FUNCTION GENERATOR

RAMP AMPLITUDE

SYMMETRY

SYMM OFF

MIN MAX

DC OFFSET/RAMP FREQ. (OUT) 0V (IN)

-5V +5V

AMPLITUDE

MIN MAX

FREQUENCY

ALL OUT X 200 kHz

X0.1 X1 X10 X100 X1 X10 X100

Hz kHz

TTL OUTPUT

FM

ON OFF

AM

RAM 0-10V

MINI-LAB

MODEL 603B BWD INSTRUMENTS

POWER

ON

VOLTS

VOLTAGE/OPERATIONAL AMPLIFIER

GAIN

OP AMP

100

INPUT

OUTPUT

BI-POLAR POWER SUPPLY

0

-15V +15V

±15V 1 AMP

+5V FIXED

REGULATED POWER SUPPLY

ISOLATED TO ±200V MAX

1.5-15V 1 AMP

0

1.5-15V 1 AMP

1 AMP MAX

6-3 VAC

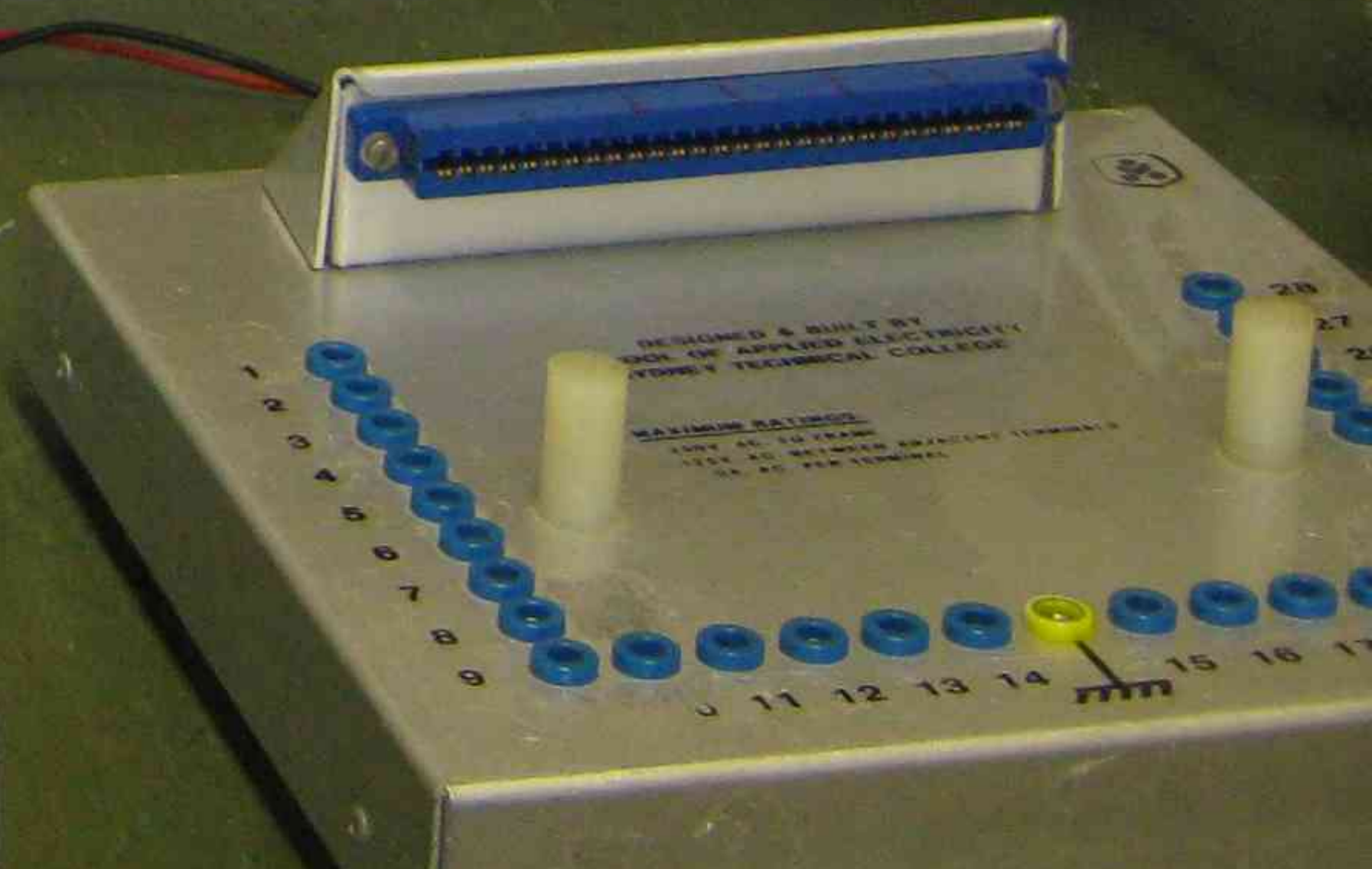
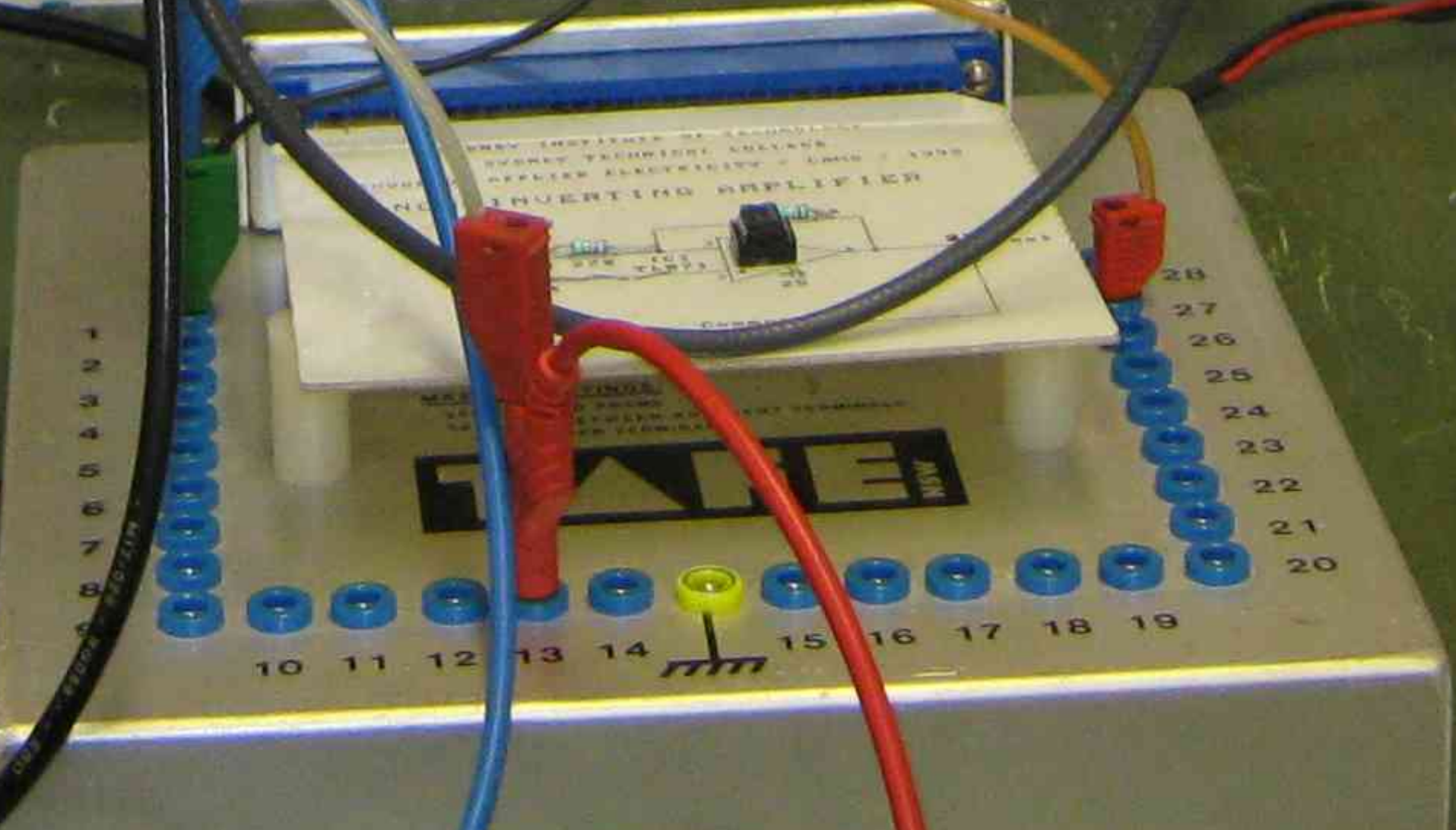
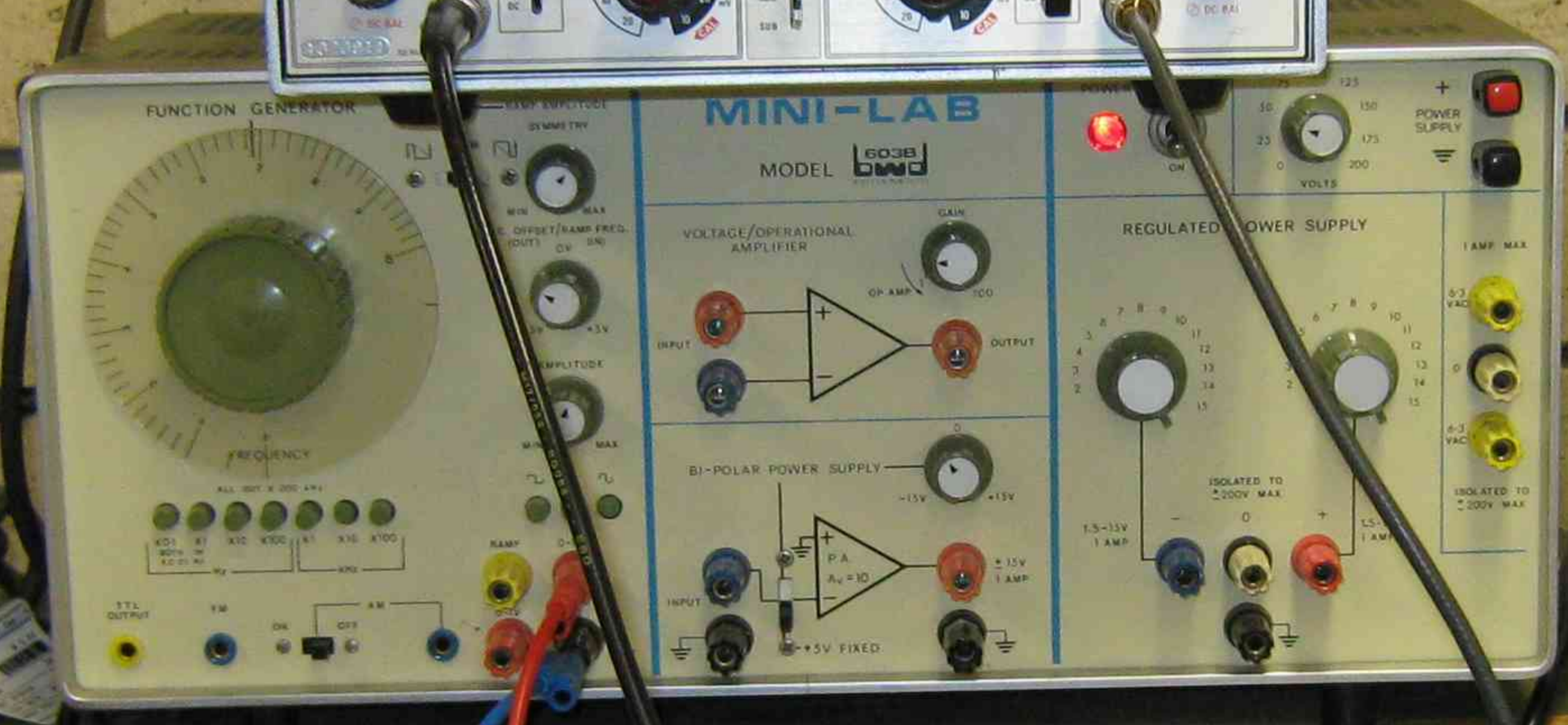
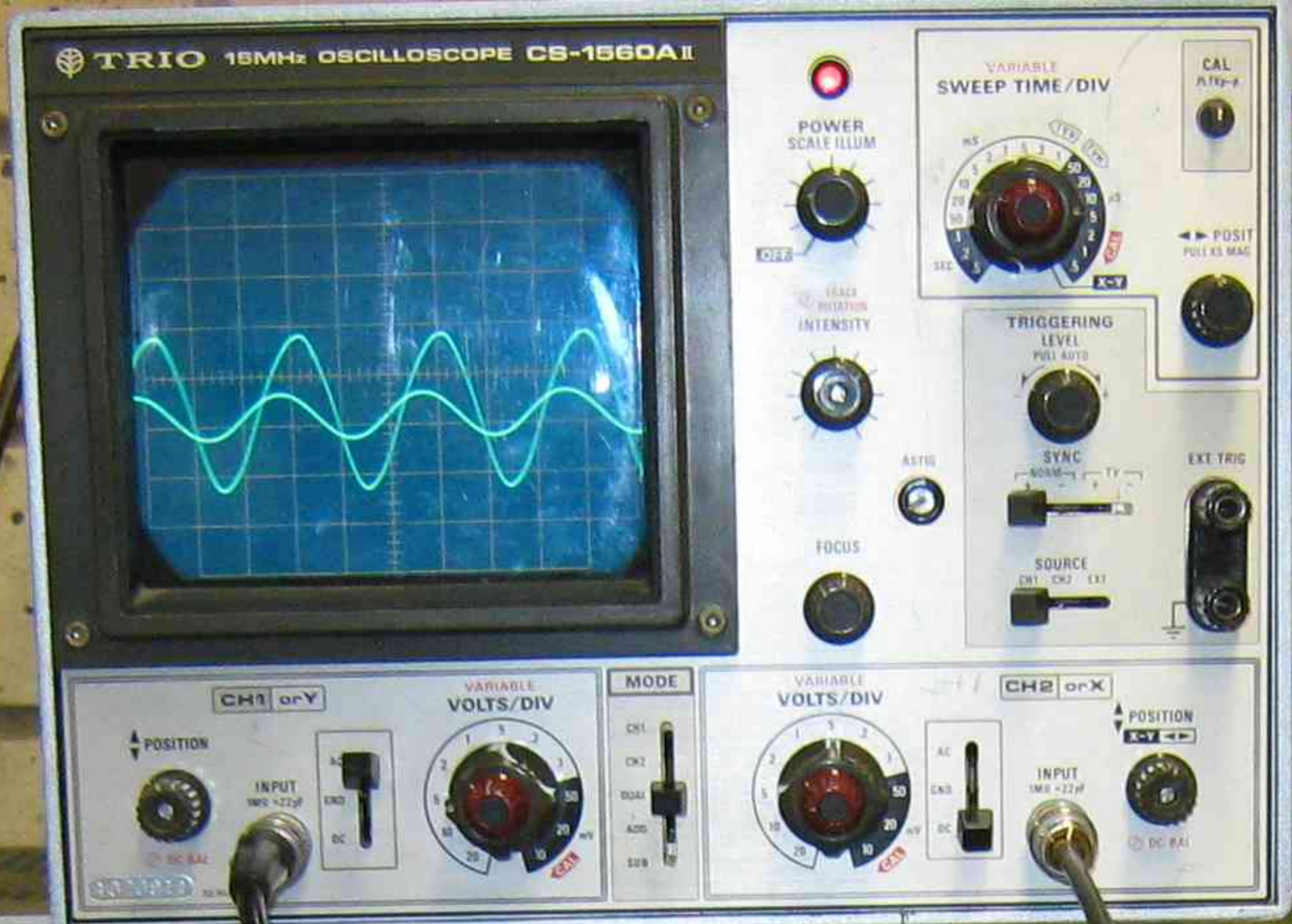
0

6-3 VAC

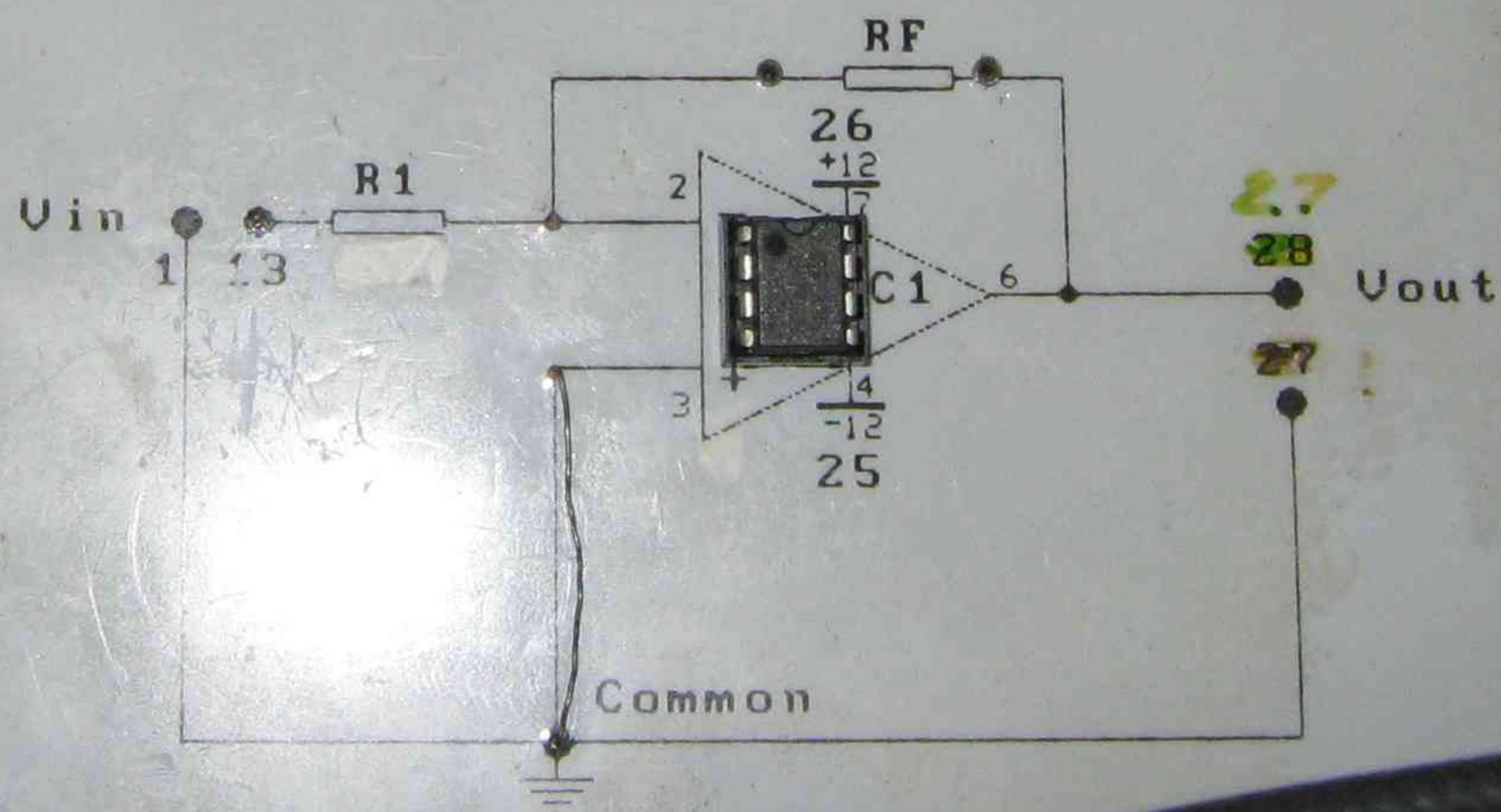
ISOLATED TO ±200V MAX

450768 155
PLIANCE 10

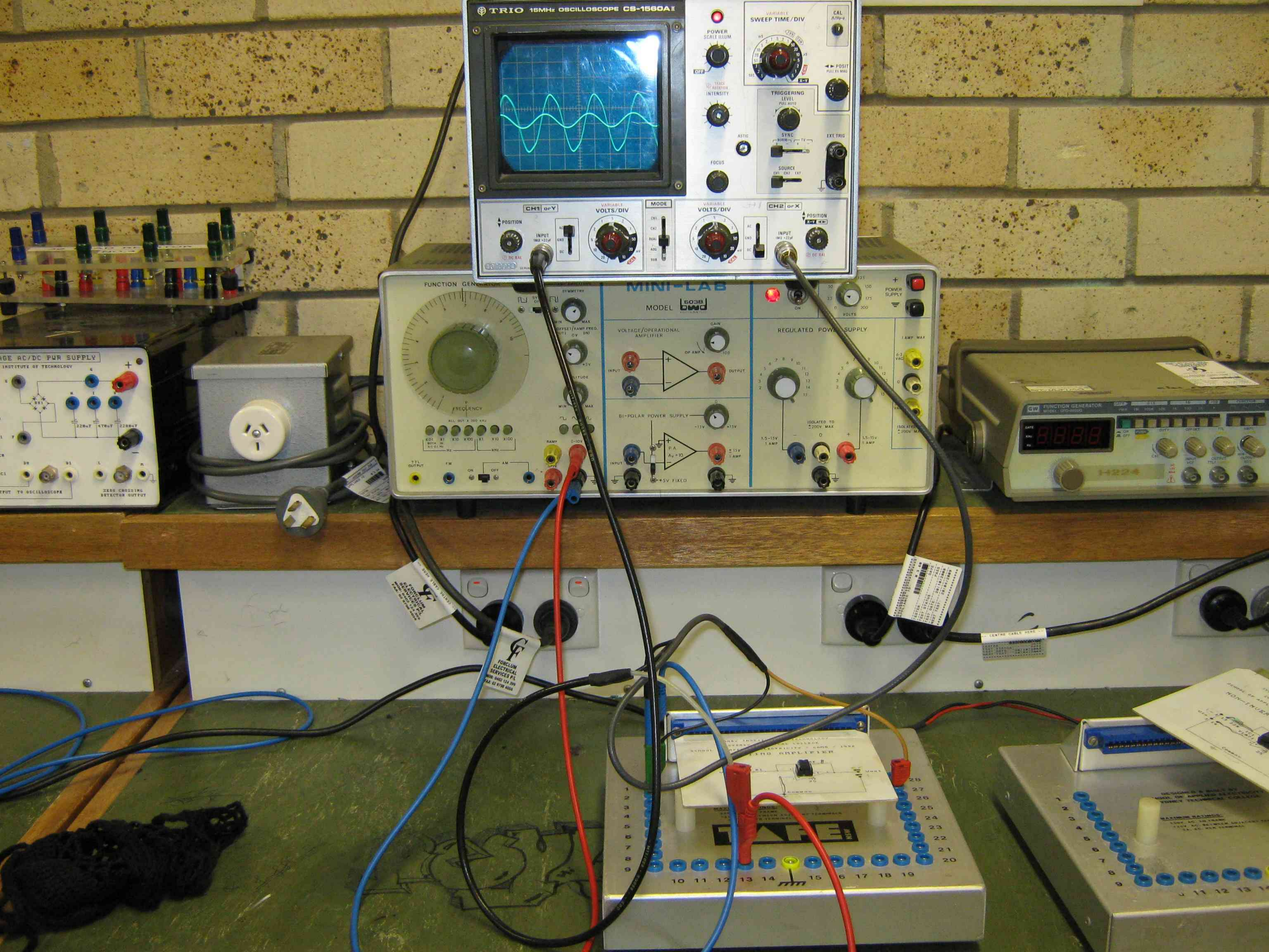
W-3 CE
STATUS DAVE
DATE 28/10/2009
PAGE 28/10/2009
NAME TO



SYDNEY INSTITUTE OF TECHNOLOGY
SYDNEY TECHNICAL COLLEGE
SCHOOL OF APPLIED ELECTRICITY / CAMS / 1992
INVERTING AMPLIFIER



- 3
- 2
- 1
- 5
- 6
- 7
- 8
- 28
- 27
- 26
- 25
- 24
- 23
- 21
- 20



TRIO 15MHz OSCILLOSCOPE CS-1560A II

POWER SCALE ILLUM. [ON/OFF]

SWEEP TIME/DIV [10, 20, 50, 100, 200, 500, 1000, 2000, 5000, 10000]

TRIGGERING LEVEL [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

TRIGGERING SOURCE [EXT TRIG, SYNC, SOURCE]

CH1 or Y [POSITION, INPUT, VARIABLE VOLTS/DIV]

MODE [CH1, CH2]

CH2 or X [POSITION, INPUT, VARIABLE VOLTS/DIV]

FUNCTION GENERATOR MINI-LAB MODEL 603B GWO

VOLTAGE/OPERATIONAL AMPLIFIER: GAIN, INPUT, OUTPUT

BI-POLAR POWER SUPPLY: -15V, +15V, 0

REGULATED POWER SUPPLY: 1.5-15V, 1 AMP

FREQUENCY: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]

POWER SUPPLY: 0, 100, 175, 200 VOLTS

FUNCTION GENERATOR MODEL H224

DIGITAL DISPLAY: 8888

CONTROL KNOBS: FREQ, AMPL, WAVEFORM

LARGE AC/DC PWR SUPPLY

INSTITUTE OF TECHNOLOGY

220VAC, 470uF, 2200uF

OUTPUT TO OSCILLOSCOPE

ZERO CROSSING DETECTOR OUTPUT

Power outlet with a three-pronged plug.

FORCLUM ELECTRICAL SERVICES PT. LTD. 14500 145th St. Surrey, BC V4W 2G7

BREADBOARD

1 2 3 4 5 6 7 8 9

10 11 12 13 14 15 16 17 18 19

20 21 22 23 24 25 26 27 28

BREADBOARD

1 2 3 4 5 6 7 8 9

10 11 12 13 14

CH1 or Y **VARIABLE VOLTS/DIV** **MODE** **VARIABLE VOLTS/DIV** **CH2 or X**

POSITION **INPUT** 1MΩ ≈ 22pF AC GND DC

CH1 CH2 DUAL ADD SUB

DC BAL 10:200:1 SE

AC GND DC **INPUT** 1MΩ ≈ 22pF POSITION X-Y

FUNCTION GENERATOR

RAMP AMPLITUDE SYMMETRY

SYMM OFF

MIN MAX D.C. OFFSET/RAMP FREQ. (OUT) 0V (IN)

AMPLITUDE

MIN MAX

FREQUENCY

ALL OUT X 200 kHz

X0.1 X1 X10 X100 X1 X10 X100

BOTH IN X0.01 Hz kHz

TTL OUTPUT FM ON OFF AM

MINI-LAB

MODEL **603B** **BWD** INSTRUMENTS

VOLTAGE/OPERATIONAL AMPLIFIER

GAIN

OP AMP

INPUT OUTPUT

BI-POLAR POWER SUPPLY

0 -15V +15V

INPUT

P.A. $A_v = 10$

+5V FIXED

REGULATED POWER SUPPLY

POWER SUPPLY

ON

0 25 75 150 200 VOLTS

1 AMP MAX

6.3 VAC

0 6.3 VAC

ISOLATED TO ±200V MAX

1.5-15V 1 AMP

0

1.5-15V 1 AMP

ISOLATED TO ±200V MAX

DATE: 28/10/2008
 TIME: 10:10:00
 USER: DAVE
 PASS: 12345678
 M 3.00

TRIO 15MHz OSCILLOSCOPE CS-1560A II



POWER SCALE ILLUM

OFF

VARIABLE SWEEP TIME/DIV

mS 2 1 .5 2 5 10 20 50 1 2 5 SEC

CAL μ Vp-p

POSIT PULL X5 MAG

TRACE ROTATION INTENSITY

ASTIG

TRIGGERING LEVEL PULL AUTO

SYNC NORM TV

EXT. TRIG

FOCUS

SOURCE CH1 CH2 EXT

CH1 or Y

POSITION

4020049

INPUT 1M Ω \approx 22pF

AC GND DC

VARIABLE VOLTS/DIV

1 .5 2 5 10 20 50 mV CAL

MODE

CH1 CH2 DUAL ADD SUB

VARIABLE VOLTS/DIV

1 .5 2 5 10 20 50 mV CAL

CH2 or X

AC GND DC

INPUT 1M Ω \approx 22pF

POSITION X-Y

DC

DC OFFSET RAMP FREQ. (OUT) 0V (IN)

VOLTAGE/OPERATIONAL AMPLIFIER

GAIN

OP AMP

REGULATED POWER SUPPLY

1 AMP MAX

6-3 VAC

Integrated circuit

Small scale Integration (S.S.I) - A few logic gates.

Medium scale Integrated circuit (MSI) - A complete integrated circuit counter

Large scale Integrated circuit (LSI) - more than 10,000 individual transistors on ~~silicon~~ single silicon chip.

mode of operation

Sequence of operation → computer can perform a number of basic operations called machine instructions. which the user selects and orders in a way which solves a particular problem. This sequential list of operation is referred to as a program

A digital computer utilises the very high speed of execution of each machine instruction usually a few micro seconds by having the required sequence ~~of~~ of instructions (or) program stored within the computer itself. This is known as the stored program concept and is the fundamental difference between a basic calculator and computer system.

Input Temperature

output

Turn on heating element.

Decimal system

$$\begin{array}{cccccc}
 10^4 & 10^3 & 10^2 & 10^1 & 10^0 & \\
 4 & 9 & 5 & 3 & 6 & = 49536
 \end{array}$$

$$4 \times 10^4 + 9 \times 10^3 + 5 \times 10^2 + 3 \times 10^1 + 6 \times 10^0 = 49536$$

Binary system

$$\begin{array}{cccccc}
 2^4 & 2^3 & 2^2 & 2^1 & 2^0 & \\
 1 & 0 & 1 & 1 & 1 & = 1 \times 2^4 + 0 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 \\
 & & & & & = 23
 \end{array}$$

Hex decimal number

4 bit binary pattern

Hex a decimal symbol

0000

0001

0010

0011

0100

0101

0110

0111

1000

1001

1010

1011

1100

1101

1110

1111

0

1

2

3

4

5

6

7

8

9

A

B

C

D

E

F

0110

~~~~~

6

1101

~~~~~

D

= 6D (Hex)

1111

~~~~~

F

0010

~~~~~

2

= F2 (Hex)

1011

~~~~~

B

0100

~~~~~

4

1000

~~~~~

8

1110

~~~~~

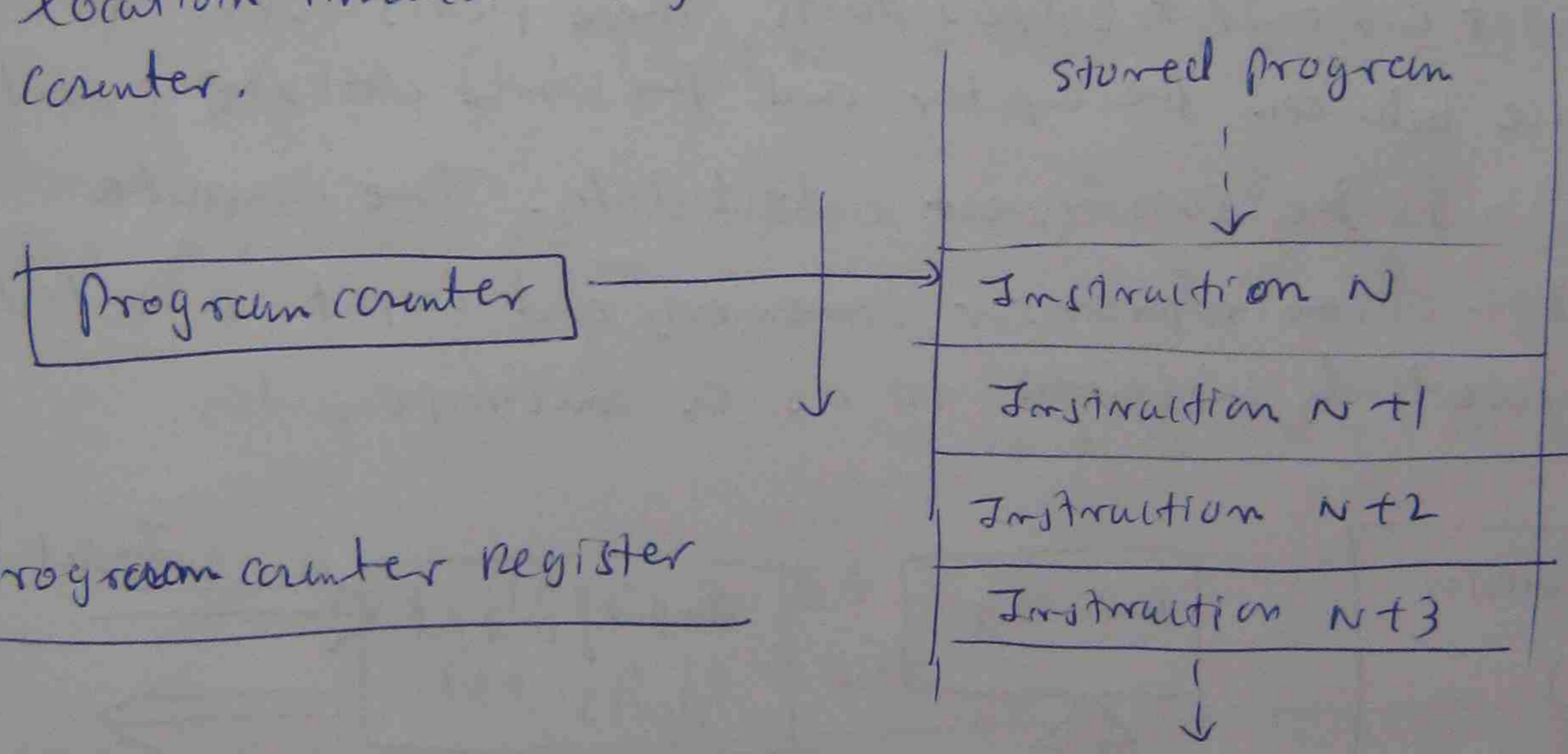
E

= B48E (Hex)

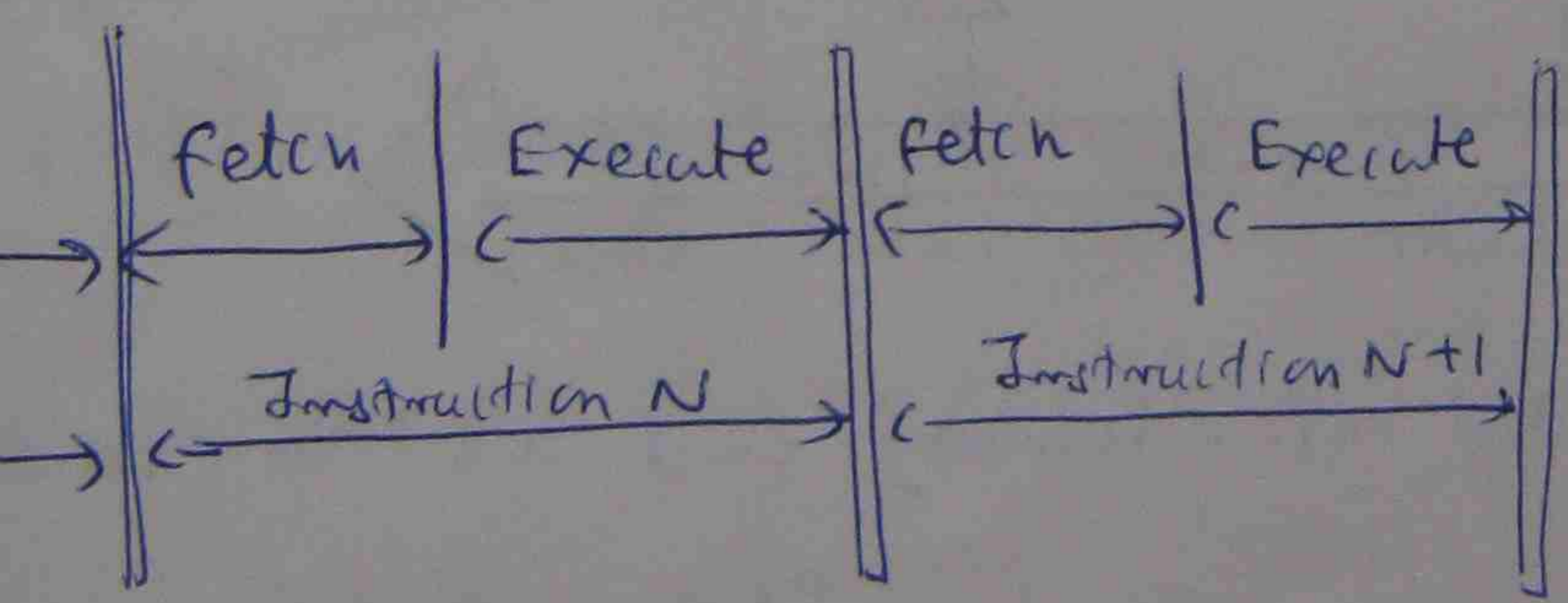
Program counter

In order to remember which program instruction is to be executed next, the microprocessor contains a register (or temporary information storage location) called the program counter (PC), the contents of which points to the next sequential instruction to be fetched and executed.

Thus during a typical instruction cycle, the next instruction to be executed is read from the memory location indicated by the contents of the program counter.



Program counter register



Problem

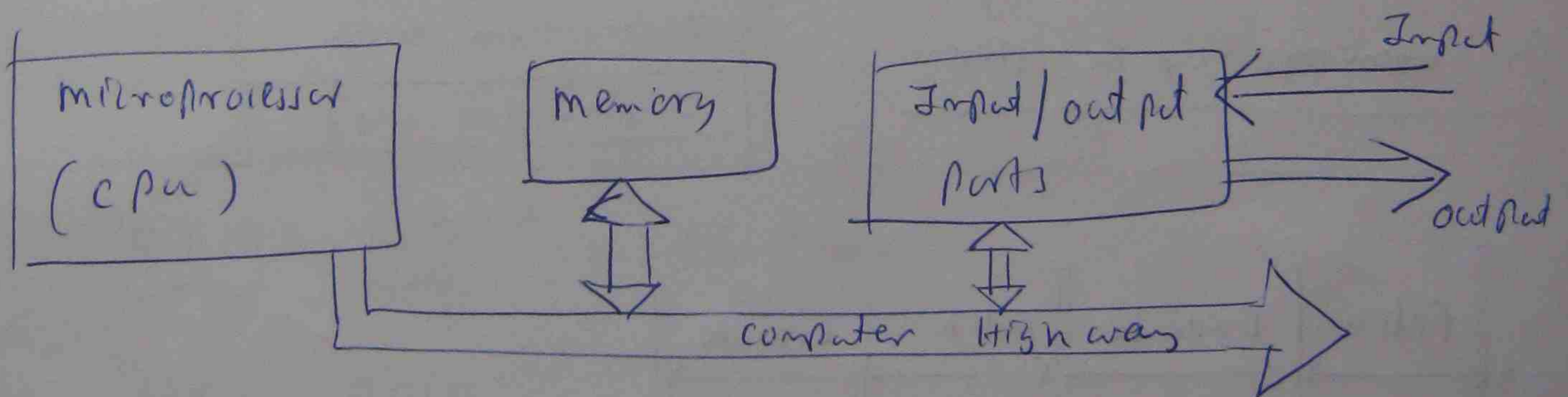
① convert the following decimal numbers into their equivalent binary numbers.

- (a) 35, (b) 67 (c) 224

Basic structure and operation

A digital computer executes a list of basic machine instructions (the program) which have been selected and ordered by the user to solve a particular task. In order to exploit ~~the~~ the intrinsic high speed of execution of each machine instruction, the program is stored within the computer.

A basic digital computer is comprised of a memory which is primarily used to hold (or) store the program, and some input and output (I/O) ports. These ports form the interface between the computer and the source of input data ~~output~~ and the subsequent output data. The complete combination of microprocessor, memory and input & output ports is collectively referred to as a microcomputer.



→ complete program is loaded into memory and is then executed.

- Phase (1)
The next instruction is fetched from memory (Fetch cycle)
- Phase (2)
The next instruction is fetched from memory; then, in the second phase (or) execution cycle. The microprocessor executes (or performs) the action specified by the instruction

35

2	35	1
2	17	1
2	8	0
2	4	0
2	2	0
	1	

Exercise

100011

(5)

To make 8 bit put 0,0

00100011

67 = 01000011

224 = 11100000

Pb (2) Convert the following binary numbers into their equivalent decimal numbers.

(a) 10111 (b) 1010101 (c) 11011011

(a) 10111 = 00010111 = $0 \times 2^7 + 0 \times 2^6 + 0 \times 2^5 + 1 \times 2^4$
 $+ 0 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 1 \times 2^0$
 $= 16 + 4 + 2 + 1 = 23$

Similarly 1010101 = 85

11011011 = 219

Pb (3) Convert the following decimal numbers into their equivalent hexadecimal number.

(a) 27 (b) 96 (c) 3334

27 =

2	27	1
2	13	1
2	6	0
2	3	1
	1	

11011 \Rightarrow $\frac{00011011}{\downarrow \quad \downarrow}$
 $= 1B$

(b) $96 \Rightarrow$

2	96-0
2	48-0
2	24-0
2	12-0
2	6-0
2	3-1
	1

(6)
 $1100000 = 0110,0000$
 ↓ ↓
 6 0
 60 (Hex)

(c)

$3334 \Rightarrow \underline{110,100000110} = D06 \text{ (Hex)}$
 ↓ ↓ ↓
 D 0 6

Pb(4)

convert the following hexadecimal numbers into their equivalent decimal numbers.

(a) D3 (b) 2F (c) 2D9E

(a) $D3 = 1101,0011 = 1 \times 2^7 + 1 \times 2^6 + 0 \times 2^5 + 1 \times 2^4 + 0 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 = 211$

(b) $2F = 0010,1111 = 0 \times 2^7 + 0 \times 2^6 + 1 \times 2^5 + 0 \times 2^4 + 1 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 = 47$

(c) $2D9E = 0010,1101,1001,1110$
 $= 0 \times 2^{15} + 0 \times 2^{14} + 1 \times 2^{13} + 0 \times 2^{12} + 1 \times 2^{11} + 1 \times 2^{10} + 0 \times 2^9 + 1 \times 2^8 + 1 \times 2^7 + 0 \times 2^6 + 0 \times 2^5 + 1 \times 2^4 + 1 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 0 \times 2^0$
 $= 11678$

Microcomputer Architecture

Microprocessor (CPU)

Memory → Hold the stored program.

Input & output ports → Interface the micro computer to the various input & output devices controlled by it.

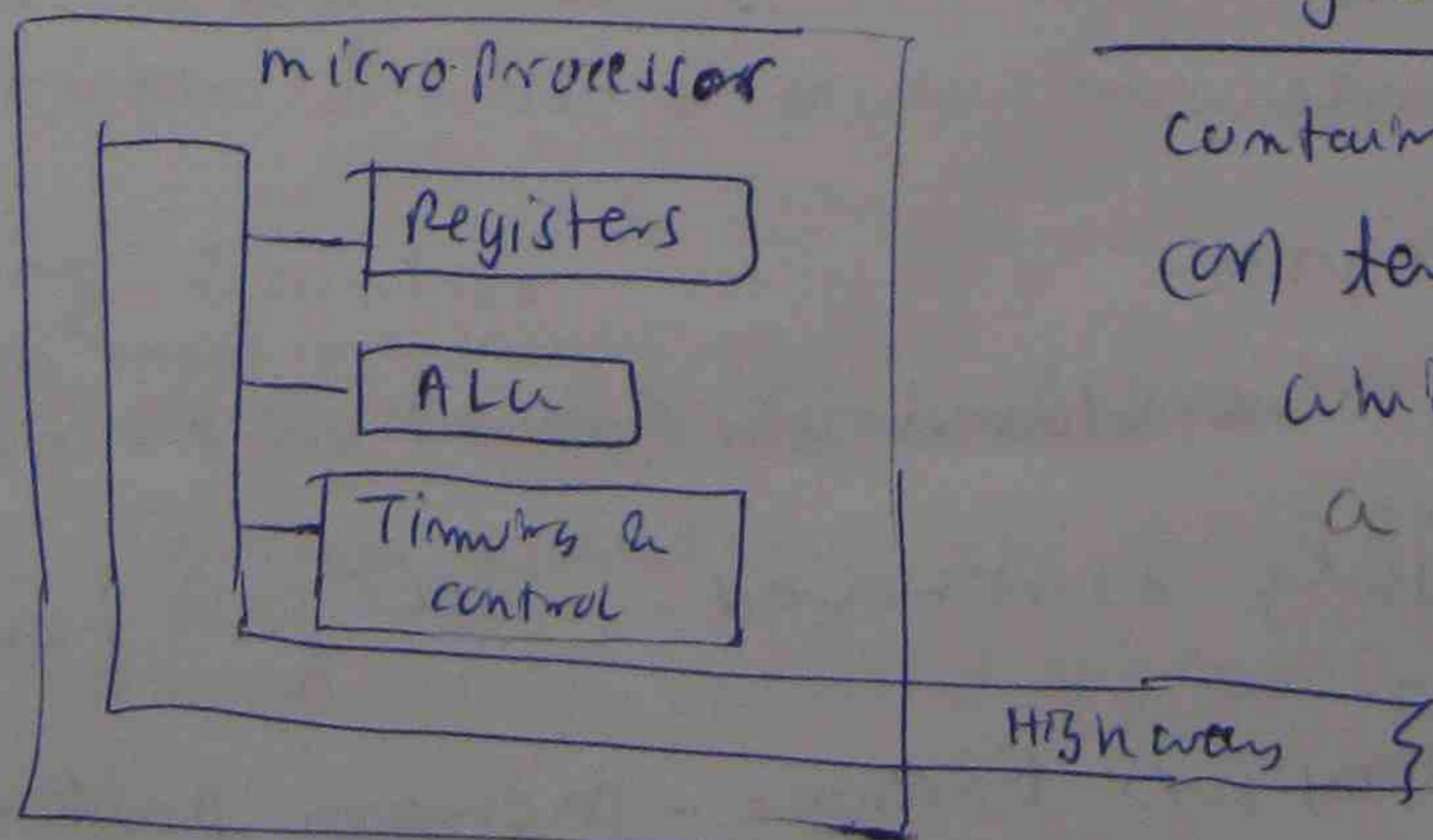
Microprocessor

Individual data byte manipulation instructions
(add, subtract)

memory transfer instructions (read data byte from memory
write data byte to memory)

Information is transferred between external devices and the computer system via the input & output ports.

The microprocessor has machine instructions to both read (input) data from a specified port and to write (output) data to a port.

Basic microprocessorRegister

contains a number of registers
(or) temporary storage elements
which can hold or store
a single byte or word

(ALU) Arithmetic Logic Unit - Performs the actual data manipulation.

Timing & control section - co-ordinates the internal operation of the ALU and registers so that the desired action specified by an instruction is performed.

The micro processor communicates with the memory, both to obtain the individual instructions which make up the program and to access & store data and to transfer data to and from input and output using a highway (or) bus.

Memory

Locations \leftrightarrow address. - Each location contains a binary pattern with a number of bits corresponding to the word length of the computer (typically 8 bits).

Content - The binary pattern stored at an address.

Memory \rightarrow RAM - Random access memory

\rightarrow ROM - Read only memory.

ROM - Fixed information during manufacture (or) by the user and consequently can only be operated in a read only mode.

- Hold fixed program

- Non volatile. The stored information is not lost when power supply is removed. Low cost / unit

Erasable Programmable ROMs (or) EPROM - Memory pattern can be changed by the user in a controlled manner.

Microcomputer ArchitectureFunctional units of a micro computer

Microprocessor - CPU

The memory - It is used to hold the stored program.

Input/output - To interface the micro computer to the various ports
input and output devices controlled by it.

Microprocessor

- Execute a number of basic machine instructions.
- Data byte manipulation, instruction (add, subtract)
- memory transfer instruction (Read data byte, transfer between external devices and the computer system via the input and output ports,)
- machine instructions
 - Read (input) data from a specified port
 - Write (output) data to a port.

Register

- contains a number of registers or temporary storage elements which can hold or store a single byte or word.

Arithmetic Logic Unit (ALU)

- Perform the actual data manipulation operations.

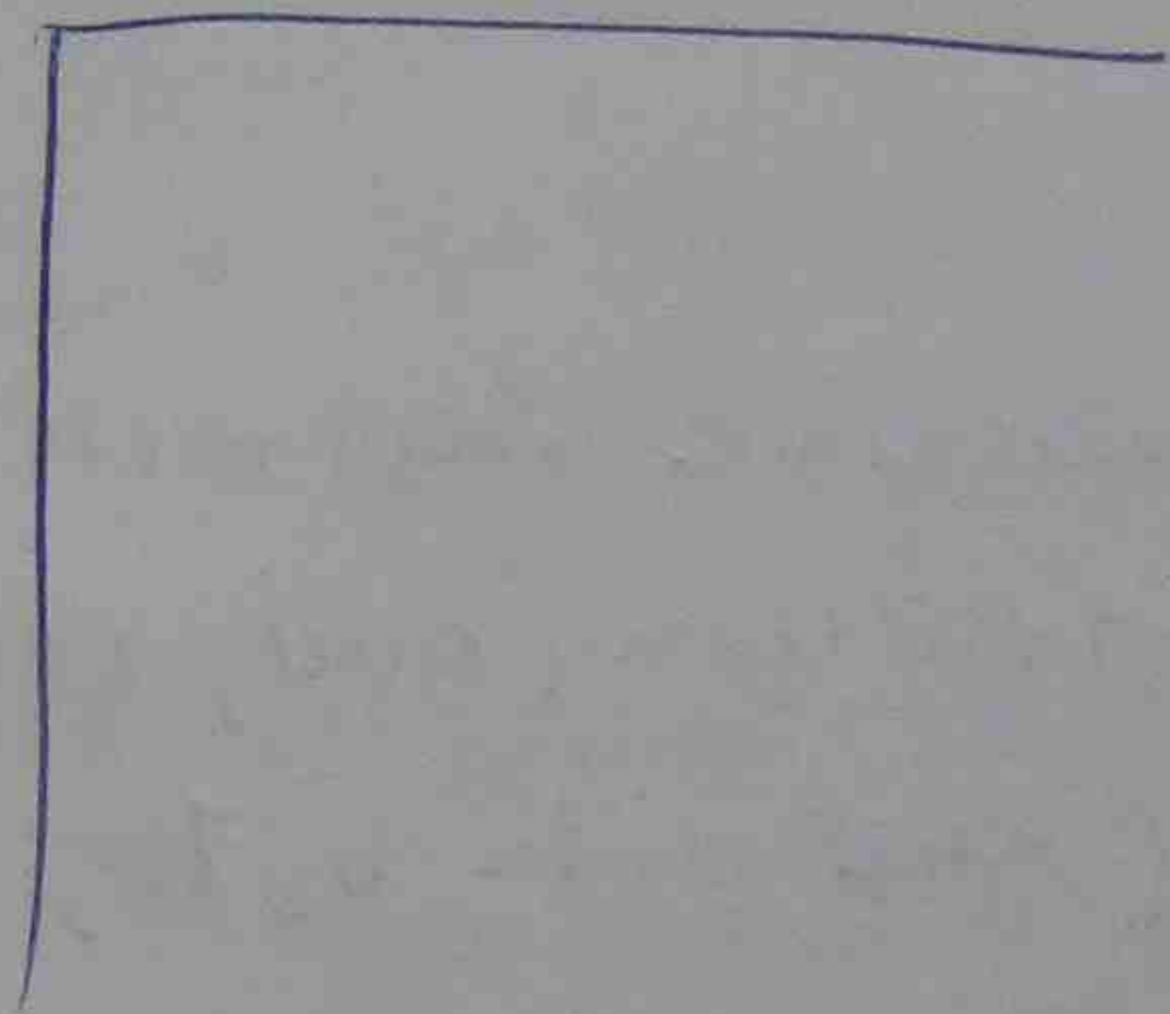
Timing Control

- co-ordinate the internal operation of the microprocessor and controls operation of the ALU and registers so that the desired action specified by an instruction is performed.

(16) (2)

Microprocessor communication

- The microprocessor communicates with the memory both to obtain the individual instructions which make up the program and to access and store data.
- To transfer data to and from input and output ports using a highway (or) bus.



Erase

(9)

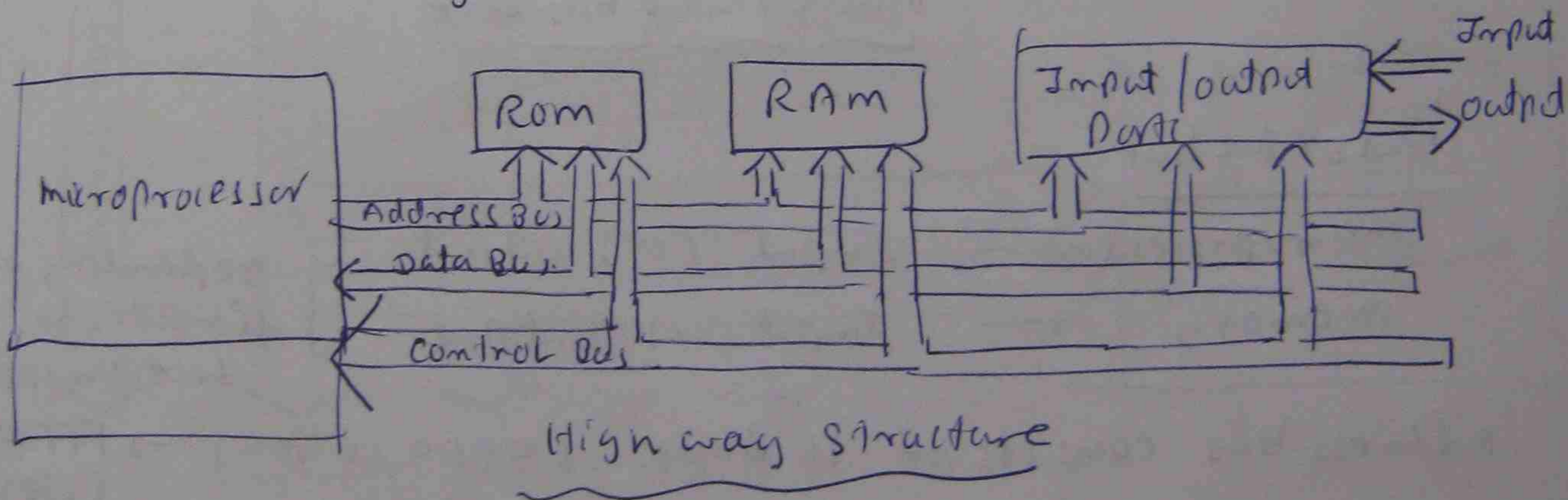
- Exposure to intense ultra violet light
- Applying voltage to ~~specific~~ specific pins on the integrated circuit.

Highway Structure

Data bus - carries the data associated with a memory or input/output transfer & typically 8 bit wide

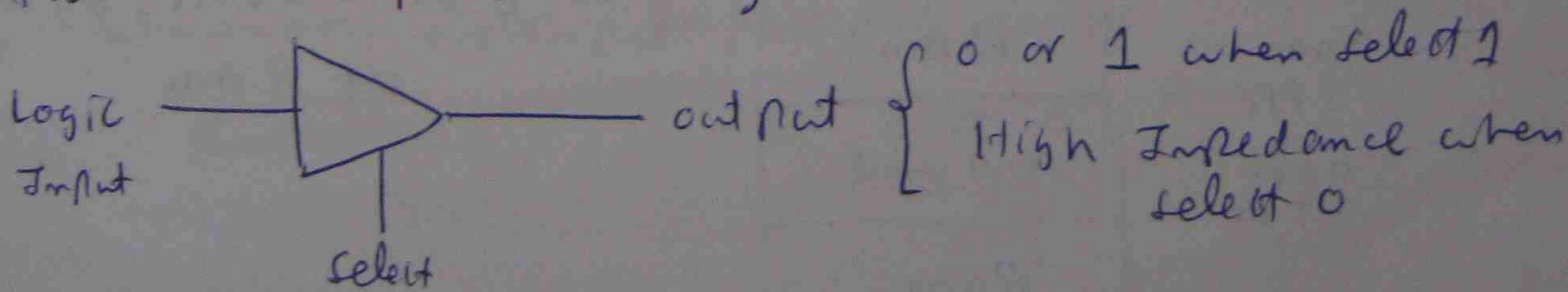
Address bus - To specify the memory location (or) input/output port involved in a transfer

Control bus - made up of the various control lines generated by the microprocessor and other system components to synchronise transfer.

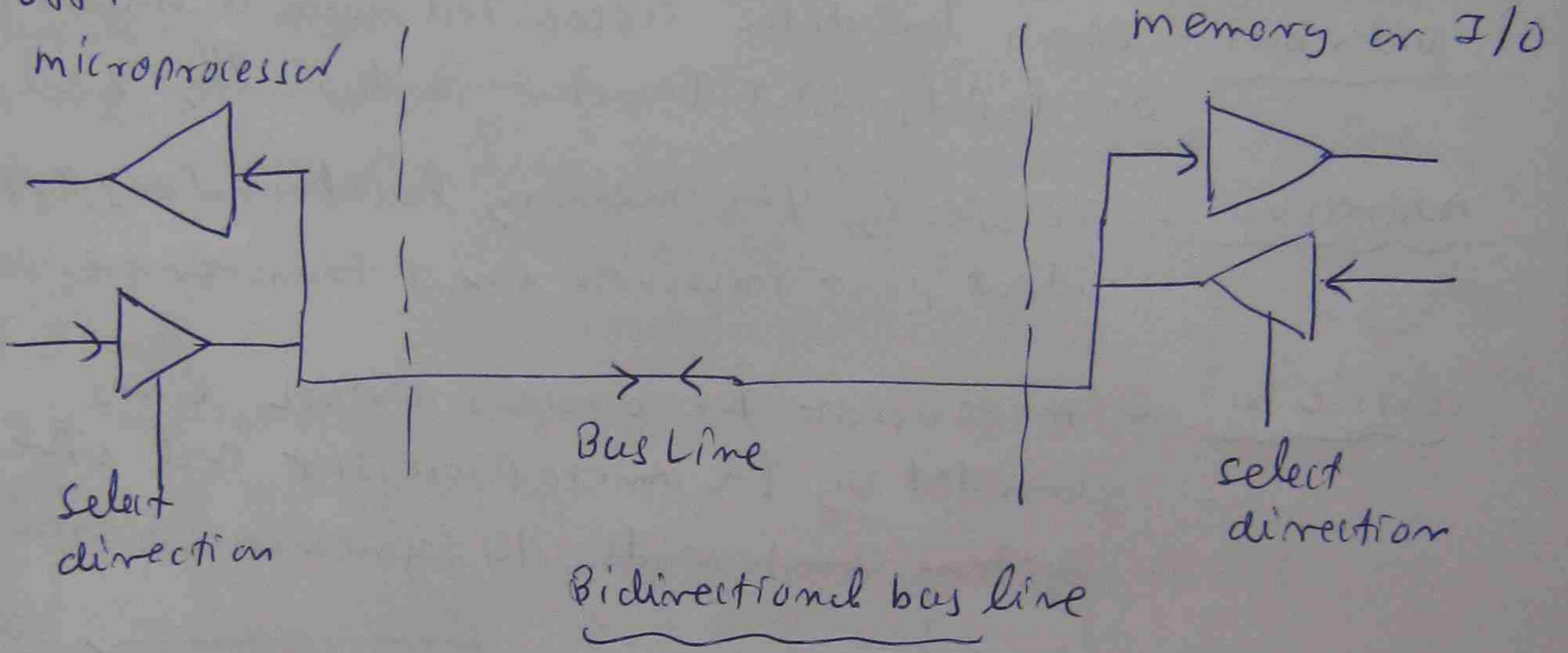


Data bus - Bidirectional data flow

The processor can write data on to the bus lines to be read by a memory device (or) it can read data from the bus presented by such a device.



It becomes possible to make a single pin a logic input and output by incorporating within the microprocessor logic output gates, a third output state in addition to normal 0 and 1 signals. This third state is a high impedance condition where the output is effectively switched off.

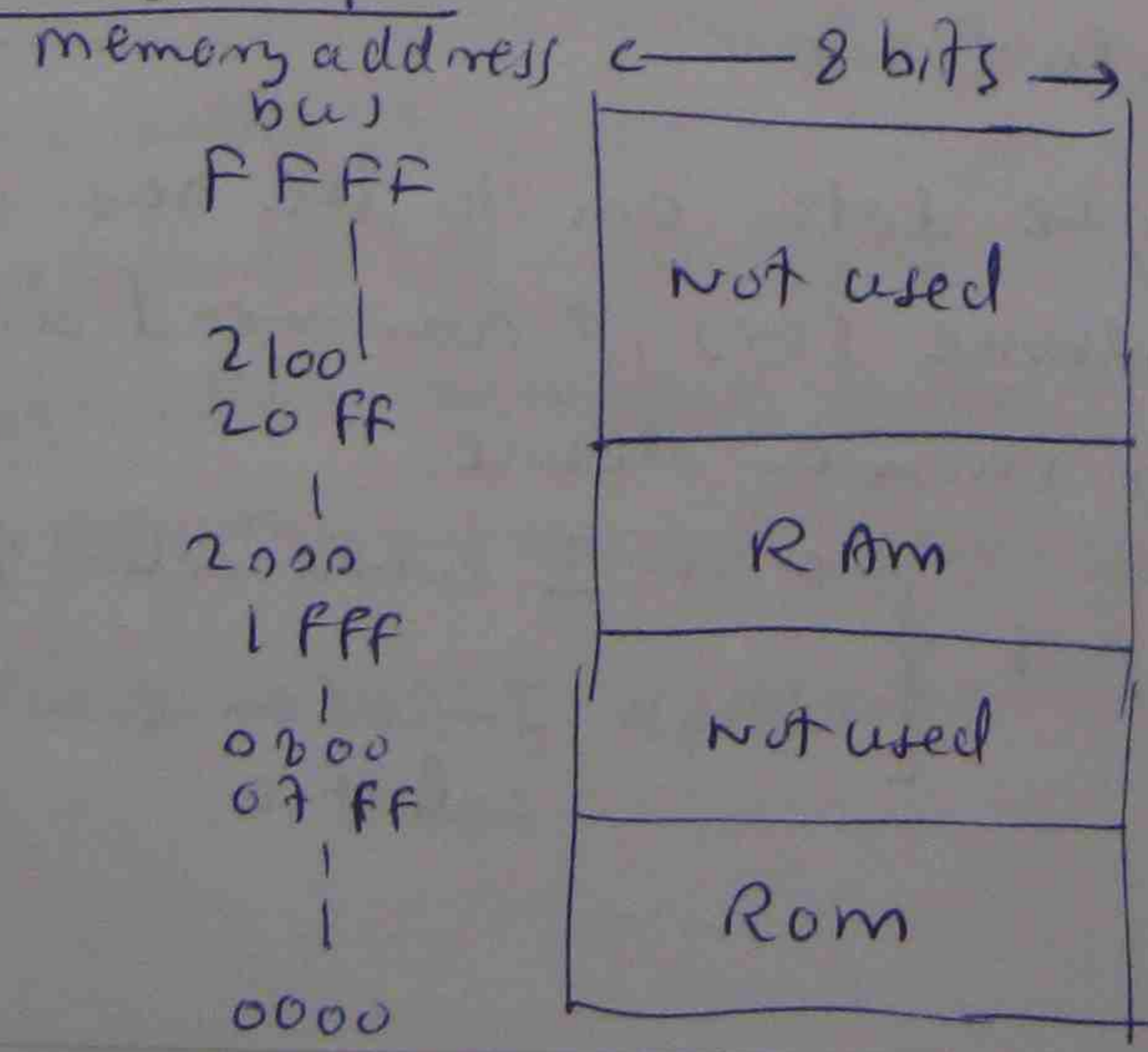


End of the bus

microprocessor — Input (or) output } depending on direction
 memory — Input (or) output } direction control

Address bus consists of 16 lines. (0000) (1hex) → FFFF (1hex)

memory map



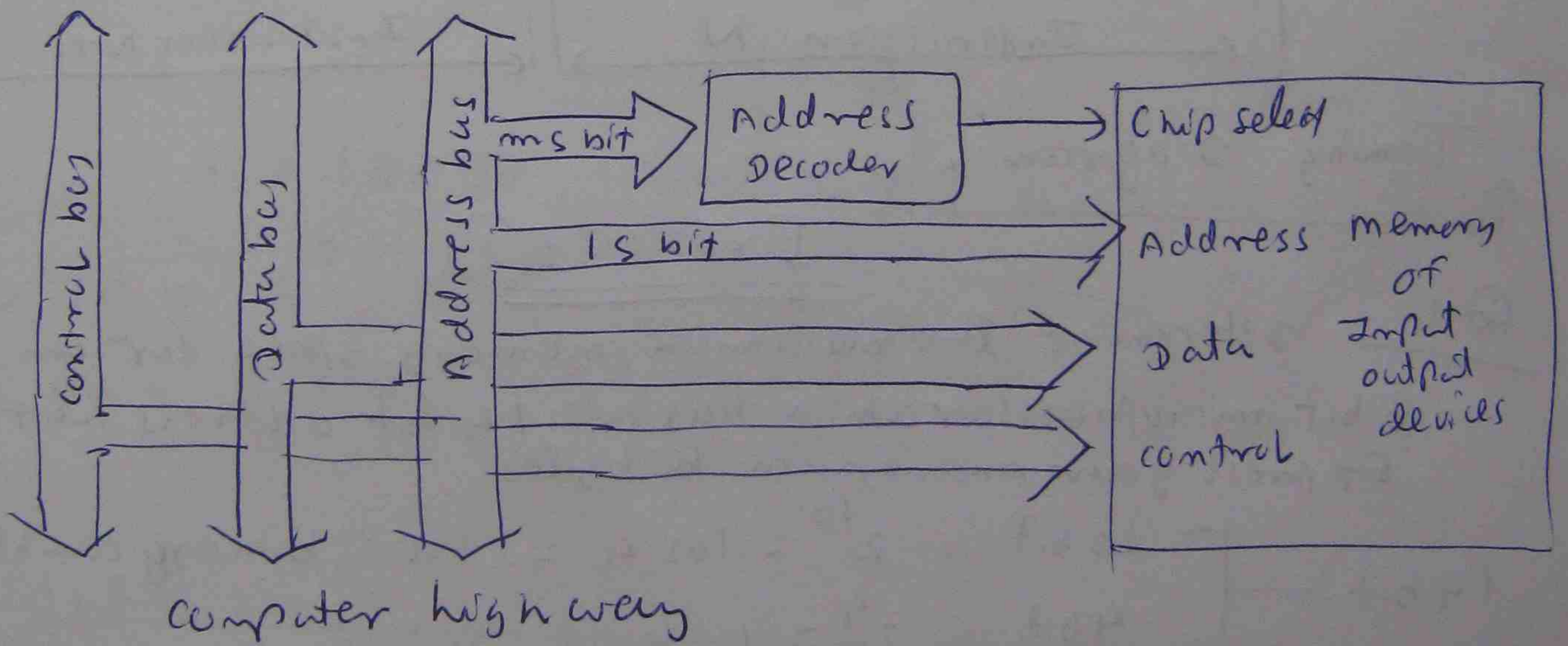
2K ⇒ 0000 → 07FF
 bytes of Rom

256 (2000 → 20FF) bytes of RAM

Address decoding

Since there are a number of devices connected to the computer highway - ROM and RAM chips, input / output devices etc - it's necessary to ensure that only the device intended for the data transfer responds when a request is made by the microprocessor.

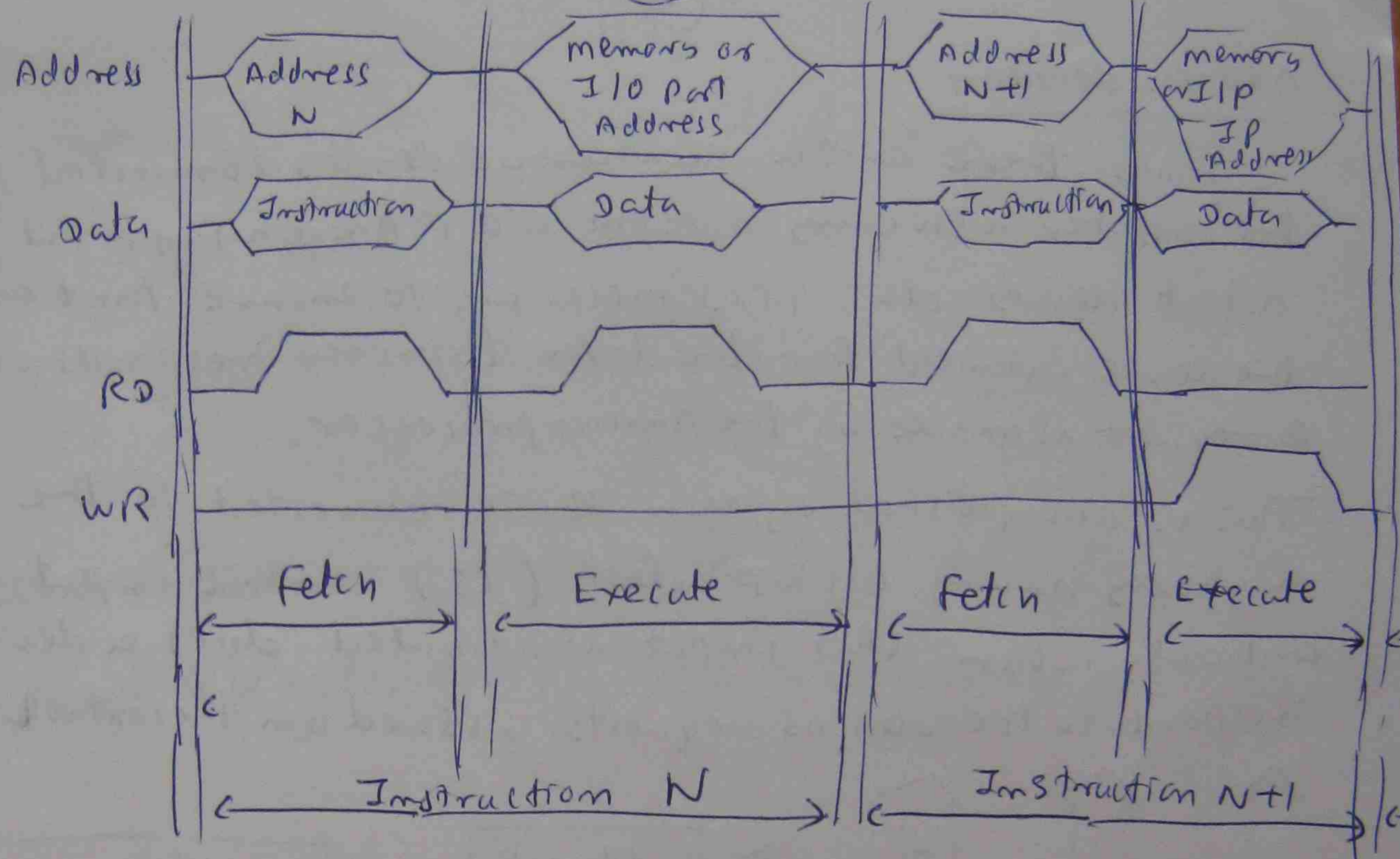
This is accomplished by each device connected to the highway having a chip select (CS) control input, and only when this input is activated does a device respond to the various requests issued on the control bus.



Bus control

The control bus incorporates the timing signals which are generated by the microprocessor to synchronise information transfers between the microprocessor & a memory or input / output port.

Read - RD, write WR



Timing Diagram

EXERCISE (2)

Q1 Determine the maximum memory space for an 8 bit microprocessor which has a 14 bit address word. Express your answer in K bytes.

14 bit — 10 bit = $2^{10} = 1024 = 1K$ binary combinations
 — 4 bit = $2^4 = 16$ binary combinations
 (0000 → 1111)

$\therefore 14 \text{ bit} = 1K \times 16 = 16K$ bytes of memory

Q2 A micro computer system requires 4K bytes of Rom and 256 bytes of RAM. Determine the start and end addresses of each memory block if the two memories are to occupy contiguous blocks of memory starting at address (0000) hex. Express your answer in hex notation.

(13)

$1\mu = 2^{10} = 1024$ locations. 10 bit
 $4\mu = 2^{12} = 4096$ locations. 12 bit
 $256 = 2^8$ locations. 8 bit

~~RAM = $2^{10} \approx 1000$~~ ~~$0000 \rightarrow 1000$~~ ~~RAM~~

~~$1000 \times 4 = 4\text{K bytes RAM}$~~ 0000 ← zero byte

1000
 \uparrow
 4×1000 4 Kbyte. Base line → 4×1000

\therefore RAM 0000
 \downarrow
 4×1000

$1\mu = 2^{10}$
 $4\mu = 4 \times 1\mu = 4 \times 2^{10} = 2^2 \times 2^{10} = 2^{12}$

8 bit = 0000 0000 → 1111 1111
 8 bit c' 256 byte F F

FFFF

not used.

I/O/OP/Port.

RAM

RAM

$\left. \begin{matrix} 2 \\ \times 2 \end{matrix} \right\} \leftarrow 4\mu = 2^{12}$

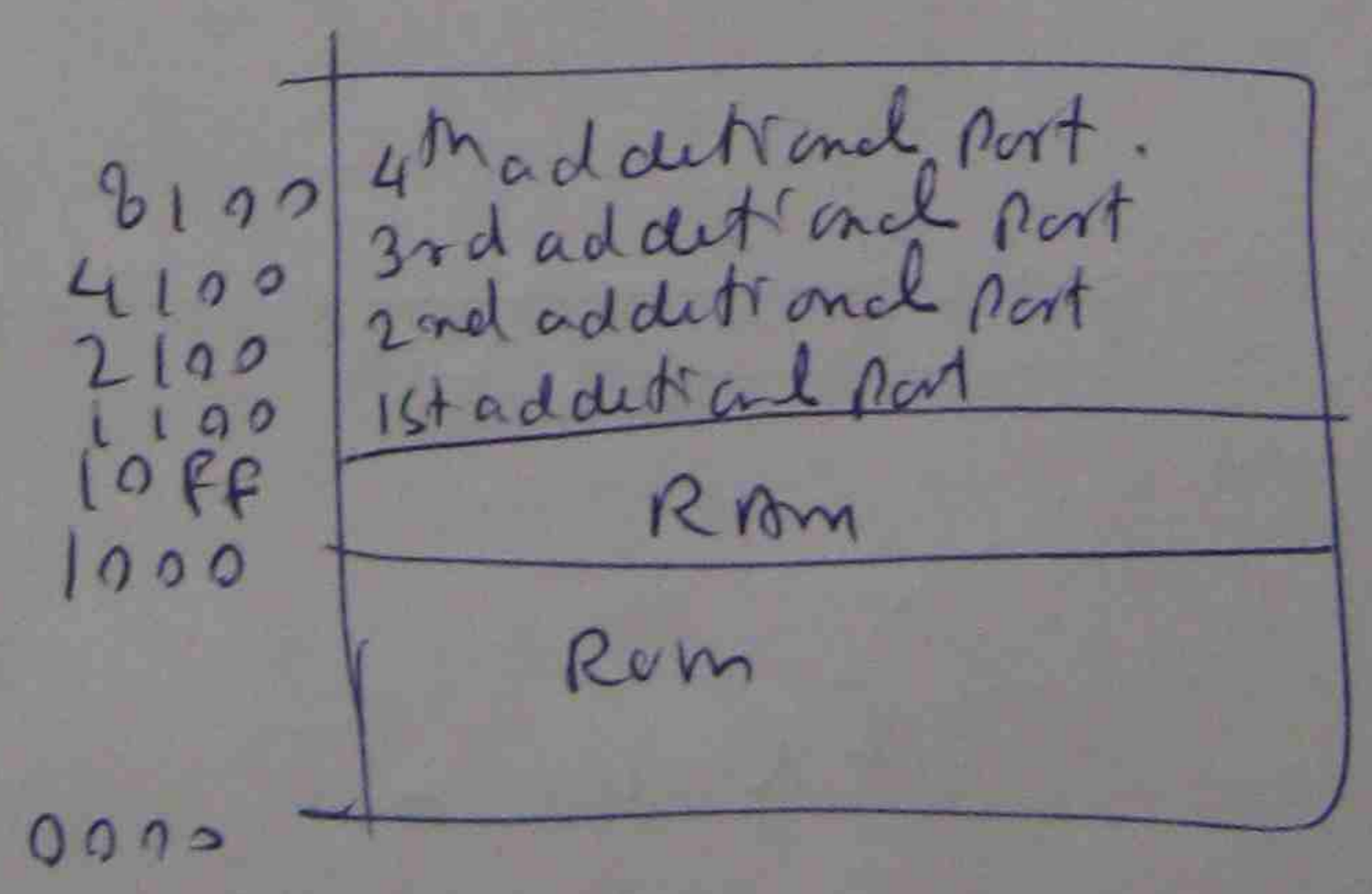
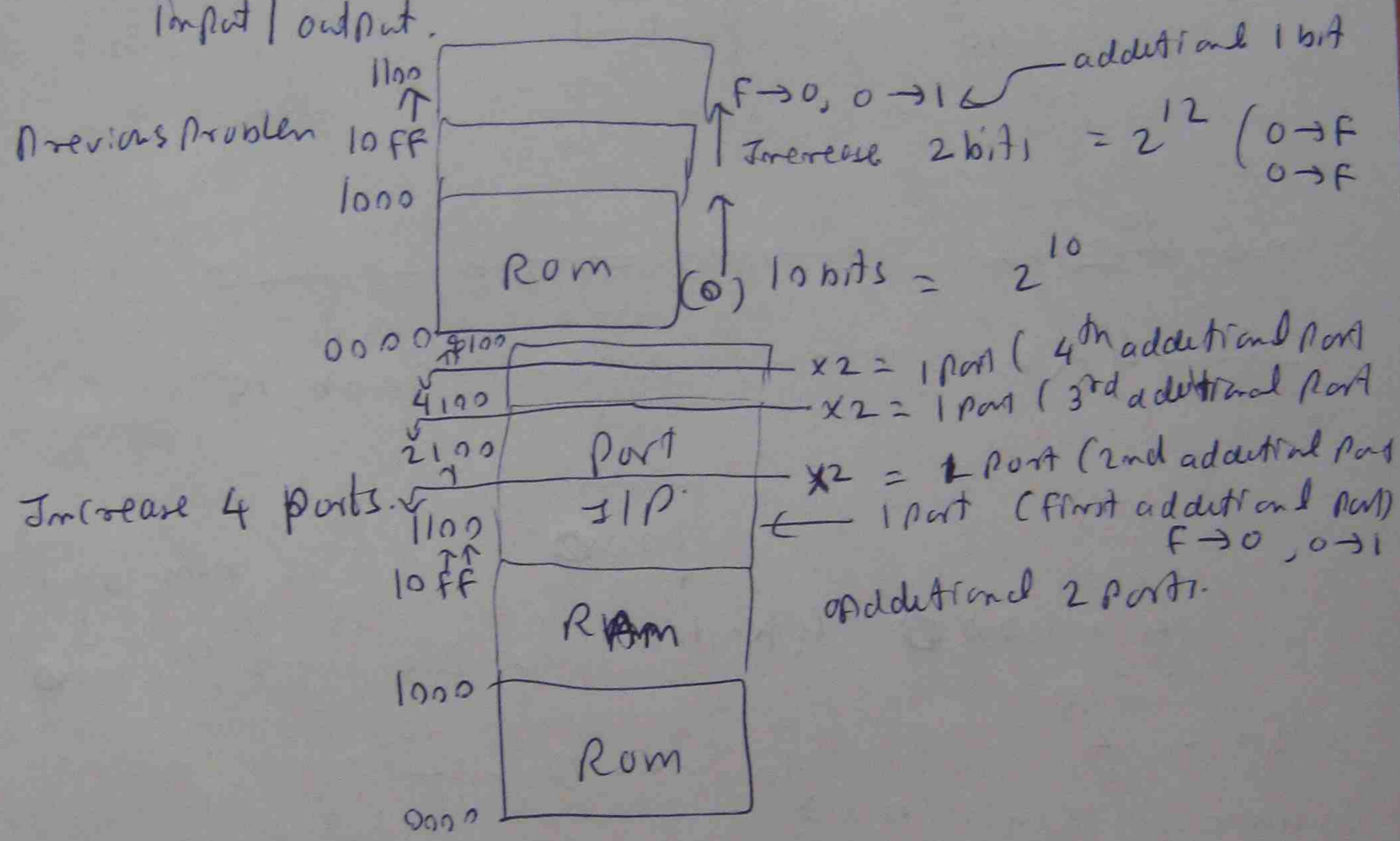
\uparrow
 $2^{10} = 1\text{K}$
 \downarrow

10 FF
 1000
 $\downarrow \downarrow \downarrow$
 0 FFF

min → max, 6 bit.

0000

Q3 If the micro computer in above question requires four additional input/output ports, define suitable addresses for the ports assuming memory mapped input/output.



Q4 A micro computer system has the following memory map.

- 0000 → 0FFF ROM
- 2000 → 21FF RAM
- 4000 → 400F I/O

Determine the amount of ROM & RAM memory and the number of I/O ports in the system.

4 bits → 1111 = 4 bit

0000 → 0FFF

2000 → 21FF

4000 → 400F

Total = 12 bit

9 bit = $2^9 = 512$ byte

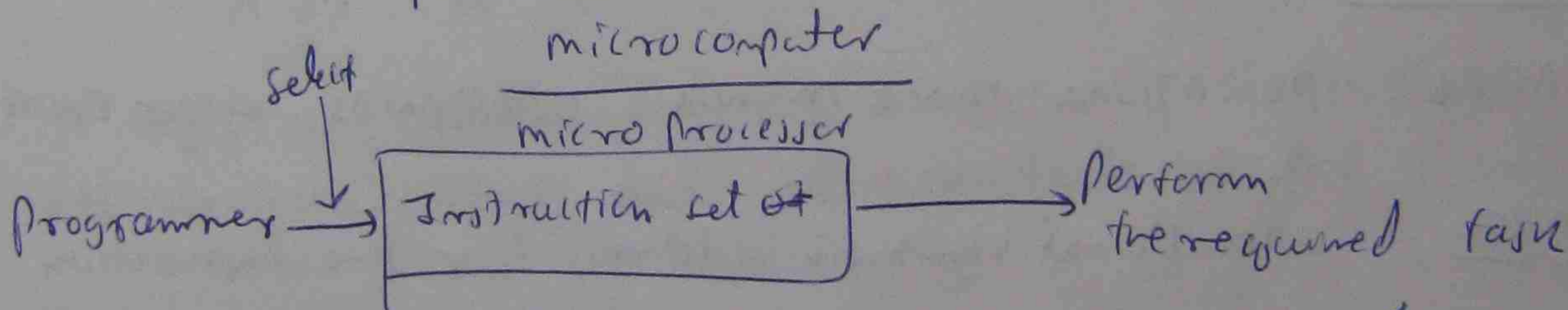
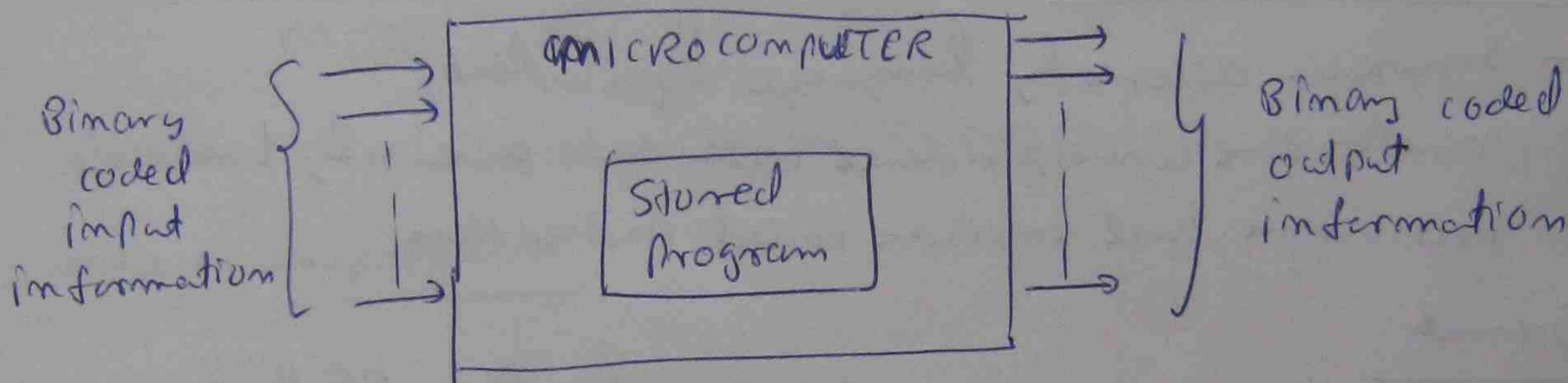
2¹² = 4K

2⁴ = 16 port

Introduction to programming and data transfer

microcomputer

- Read binary coded information from its input ports
- manipulate this information according to a program stored within its memory
- subsequently produce output information at its output ports



Program instructions.

* It is necessary to become familiar with the different types of machine instructions which a typical microprocessor executes and to investigate their effect on the total system.

Intel 8085

A	
B	C
D	E
H	L
F	Im
Stack pointer SP	
Program counter PC	

A = 8 bit arithmetic register (accumulator)

BCDE - 4 bit general purpose registers.

F = 8 bit flag register controlled by ALU

Im - 8 bit interrupt control register

HL - two 8 bit registers used to form a 16 bit memory pointer.

SP - Stack pointer register.

16 bit memory address which always points to the top of a system stack.

PC - Program counter register which contains a 16 bit memory address which points to the next instruction to be executed.

Assembly Language

- Symbolic assembly language equivalent.
- one to one correspondence between assembly language instruction and machine coded instruction.

Format

LABEL: OPERATION MNE MONIC, OPERANDS COMMENTS,

LABEL - operational symbolic address for the instruction

OPERATION MNE MONIC - Tell the programmer the specific operation to be performed

OPERANDS - A value on which this operation is to be carried out

(OR)

The memory locations where the value can be found.

COMMENTS - optional comments.
To facilitate understanding & enhance the readability of the complete program
- Does not influence the machine code resulting from the assembly instruction

Classification of instructions

- Data transfer
- Data manipulation
- Transfer of control
- Input / output
- machine control

1 Data Transfer

```
MOV A, B
```

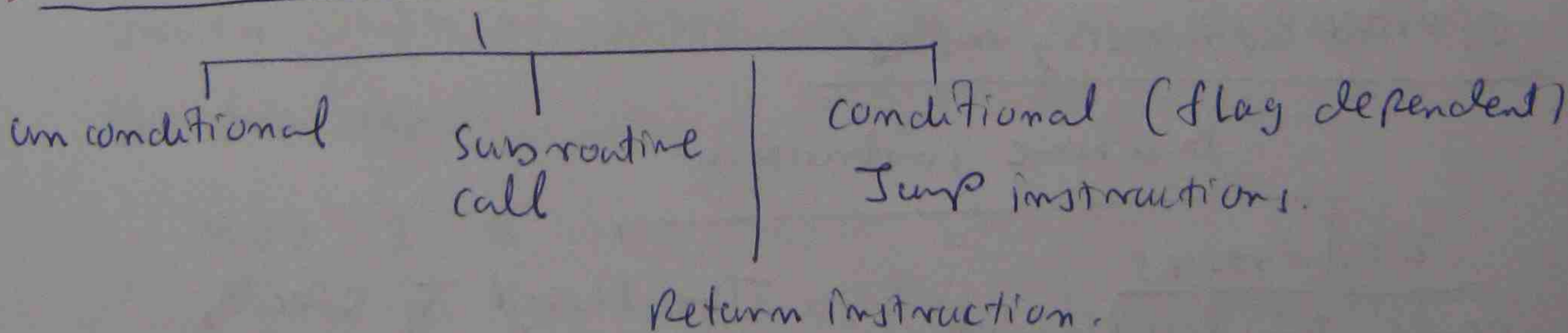
Results in B register being transferred to A register

2 Data manipulation

```
ADD A, B
```

A register (accumulator) containing the sum of its previous contents and the contents of the B register.

3 Transfer of control



```
JMP LABEL1
```

- micro processor breaks its normal mode of sequential instruction execution.
- Jump unconditionally to symbolic address LABEL1 for next instruction to be executed.

4 Input/output

move data between the various input/output ports of the system and an internal processor register - usually A-register.

OUT 05

The content of the A register being transferred to output port 05 (hex).

5 machine control

Instructions in the machine control group affect the state (or) mode of operation of the processor itself.

Interrupt, enable, disable, processor halt, no operation instructions.

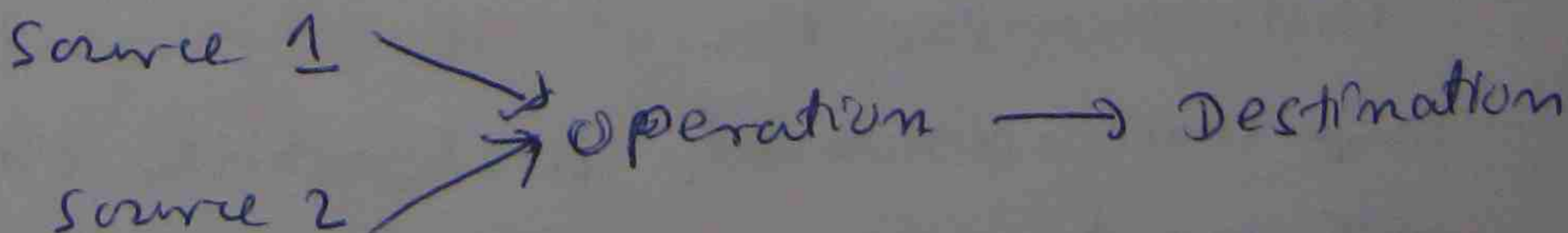
Operand addressing mode

machine instruction

2 addresses

Specify the location of the values to be manipulated
(Source addresses)

The third to specify the location where the result is to be stored
(Destination address)



Instruction addressing mode

4 main types of addressing modes

Register Addressing

Immediate Addressing

These are used primarily for data transfer and manipulation instructions which involve only the internal processor registers.

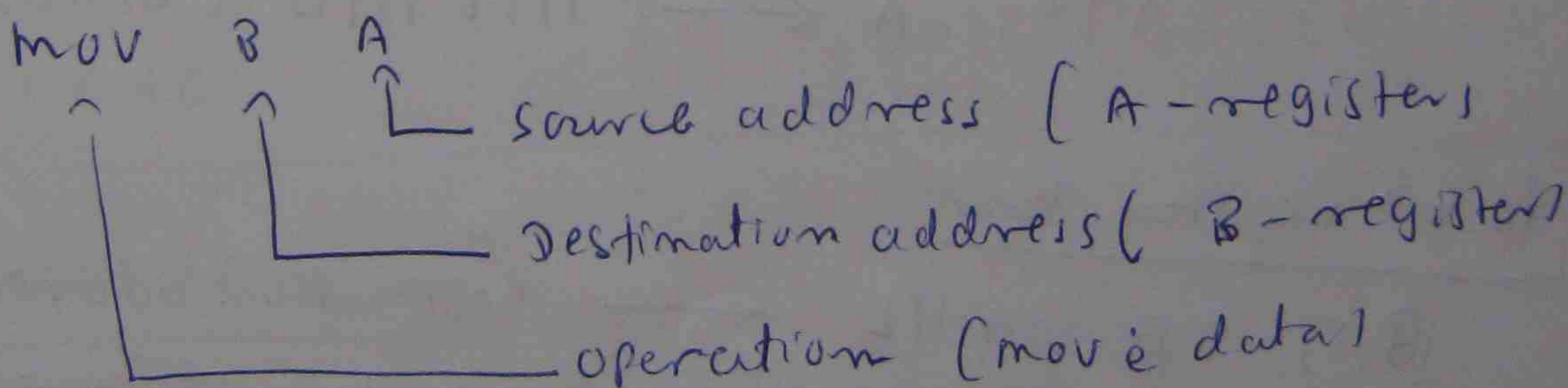
Direct (Extended) Addressing

Register Indirect Addressing

These are used primarily for data transfer and manipulation instructions which involve the system memory.

Register Addressing

- move data between the internal processor registers.
- The instruction source and destination addresses specify which of these registers are involved in the transfer.



This results in the contents of the A register being transferred to the B-register. The contents of the A-register remain unchanged. $(B) \leftarrow (A)$

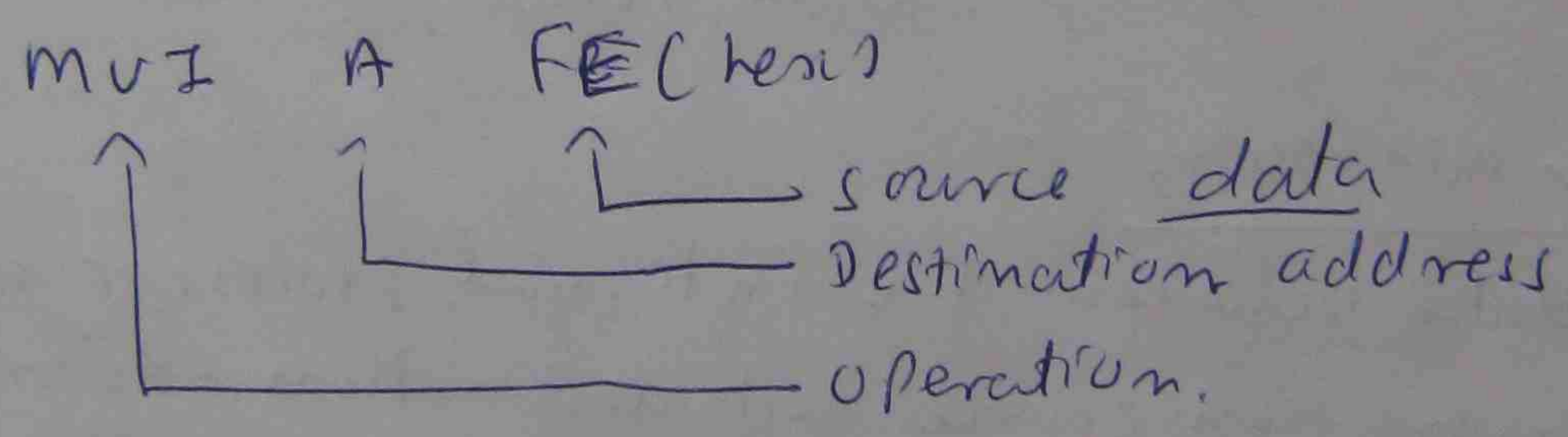
Data transfer involving 16 bit register pairs

XCHG
(DE) ↔ (HL)

The contents of register pair DE being exchanged with the contents of register pair HL.

Immediate Addressing

The source address does not specify a register (or) memory location but instead the actual source data is contained within the instruction itself, and is therefore immediately available.



The data value FE(hex) being transferred to the A register

$$(A) \leftarrow FE(hex)$$

$$(A) \leftarrow 11111110 \text{ (binary)}$$

16 bit register pair

BC, DE or HL — destination addresses.

These instructions require two bytes of immediate data

LXI H, 802D { 16 bit register pair HL being loaded with immediate data 802D(hex)

(H)(L) ← 802D(hex)

LXI D, E627 { (D)(E) ← E627(hex) Register pair DE are loaded as pair

REGISTER DATA TRANSFER

- Prob (1)
- 1 - The program loads a value into A register using immediate addressing
 - 2 - Then loads this value into two further registers B and C using ~~the~~ register addressing.
 - 3 a/b - Finally Register Pair HL and DE are loaded with 8020 & E027 using immediate addressing. ~~Then their~~
 - 4 - Then their contents are exchanged using register addressing.

Assembly Instructions			Comments	
Mnemonic	OP1	OP2		
1 MVI	A	FE	(A) ← FE (hex)	
2 {	MOV	B	A	(B) ← (A)
	MOV	C	B	(C) ← B
3a LXI	H	8020	(H)(L) ← 8020 (hex)	
3b LXI	D	E027	(D)(E) ← E027	
4 XCHG			(D)(E) ↔ (H)(L)	

Direct Addressing

An operand may be either read from (or) written to a memory location, the address of which is specified in the instruction itself.

Load

EX LDA 20EA (OR) (A) ← (20EA)

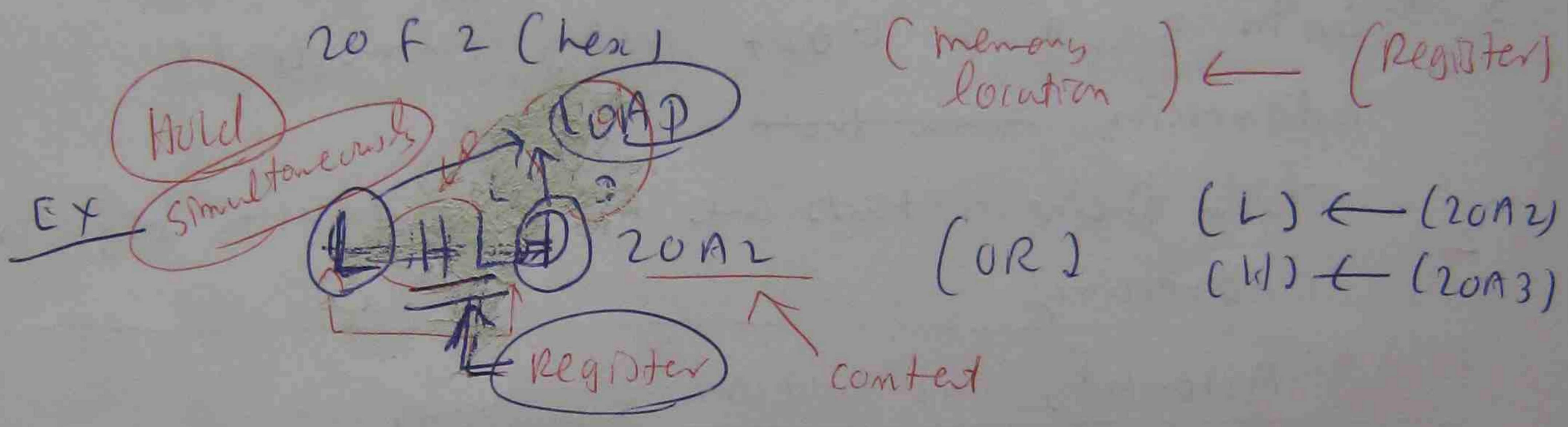
Register A loaded with content 20EA (hex)

Store

~~ALOAD~~ (Loaded) ← (content)

EX STA 20F2 (OR) (20F2) ← (A)

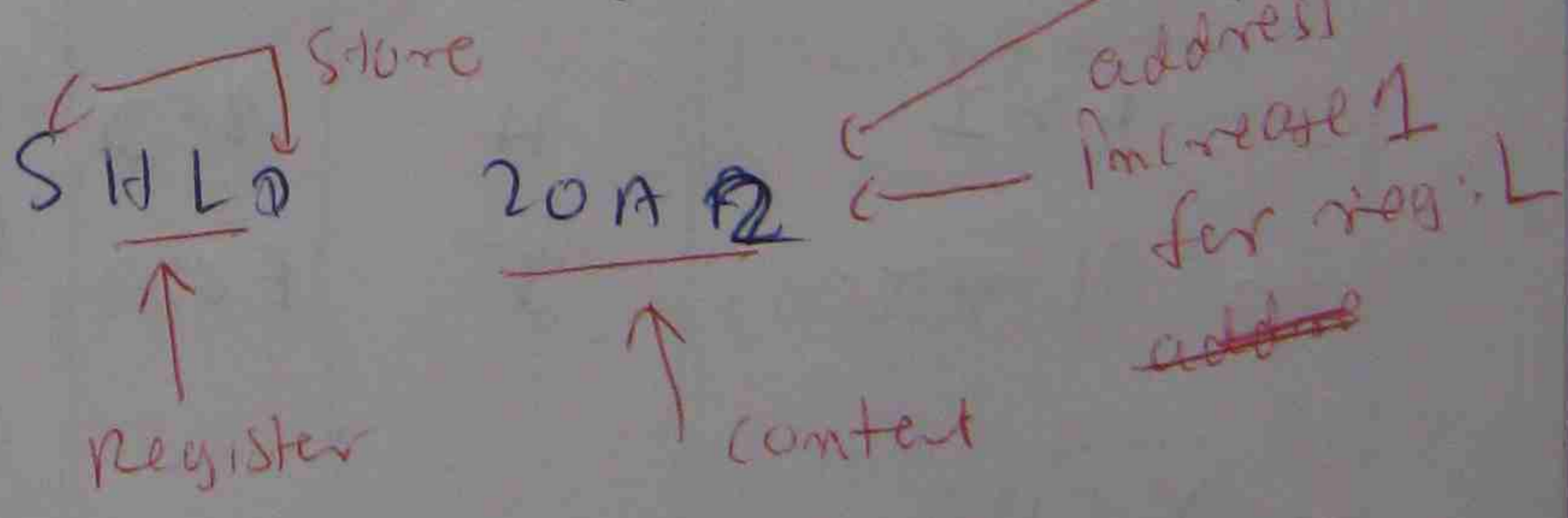
Register A being stored in memory location



Register Pair, ~~Pair 1~~ Load Hold Register 2 ~~Pair 2~~ ← content

Register pair H and L is frequently used to hold a combined 16 bit memory address.

EX Store Simultaneously



~~20AF~~ content of Register ~~(H)~~ is stored in 20A2
(L) is stored in 20A3

(L) ← (20A2)
(H) ← (20A3)

EX SHLD 20AF
(20AF) ← (L)
(20B0) ← (H)

DIRECT ADDRESSING (23)

Prob(2) - Immediate and direct addressing of register

- 1 A at FF and EE
- Data 20A2 and another consecutive memory location
- Data in A is stored at address 20A2 and another consecutive memory location.
- 2
- 3 Load register pair HL with these data
- Store the same data loaded at HL into
- 4 a two consequent memory locations.

Immediate Address (U) Store (U)
Immediate address Store (L)

	Assembly Instructions	Actions
1 (a)	MUI A, FF	(A) ← FF(hex)
2 (a)	MUI STA 20A2	(20A2) ← (A)
1 (b)	MUI A, EE	(A) ← EE(hex)
2 (b)	STA 20A3	(20A3) ← (A)
3	L ← HL D 20A2 <i>Load</i>	(L) ← (20A2) FF(hex) (H) ← (20A3) EE(hex)
4	S ← HL D 20A4 <i>Store</i>	(L) ← (20A4) (20A4) ← (L) (20A5) ← (H)

Register Indirect Addressing

Direct Addressing

Indirect addressing

more efficient method

- Only the A register may be used to store or load a value to and from memory

- data may be transferred between any of the processor registers and the system memory

- If a value were to be stored in a memory location, the B register

- operand is either read from (or) written to memory location

(1) Transfer contents from B to ~~A~~ A

- Instruction does not contain actual memory address

(2) Then store operation is performed

- Operand is either read from (or) written to the memory location, the address of which is currently stored in the register pair HL

Ex mov A m

A register being loaded with the contents of the memory location whose address is specified in register H and L

$$(A) \leftarrow (H)(L)$$

Ex movl m, FF

The value FF(hex) being stored in the memory location whose address is in register H & L $(H)(L) \leftarrow FF(hex)$

Indirect Addressing

Ph(3)

1 The memory address of A which contains data value AA is 20A0 and it is loaded into register H and L by indirect addressing mode.
 L X I H

2 Then a value is moved to memory location using register indirect addressing.
 MVI M, AA

3 The value is loaded into two further registers B & C using register indirect addressing
 MOV B, M
 MOV C, M

	Assembly Instruction	Action
1	LXI H, 20A0	(L) ← A0 (hex) (H) ← 20 (hex)
2	MVI M, AA	(H)(L) ← AA (hex) (OR) 20A0 ← AA (hex)
3	MOV B, M MOV C, M	(B) ← (20A0) (OR) (B) ← AA (hex) (C) ← (20A0) (OR) (C) ← AA (hex)

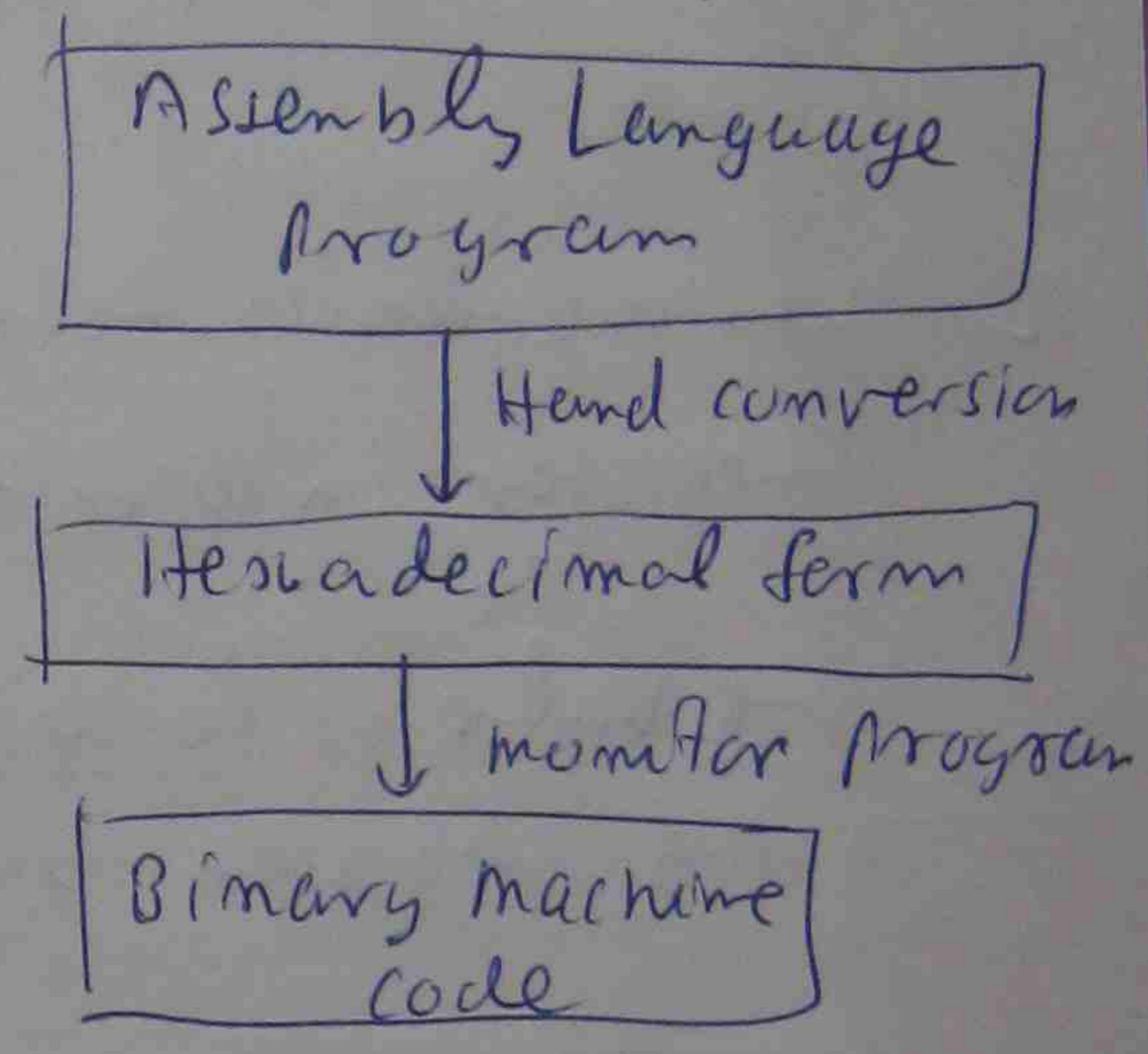
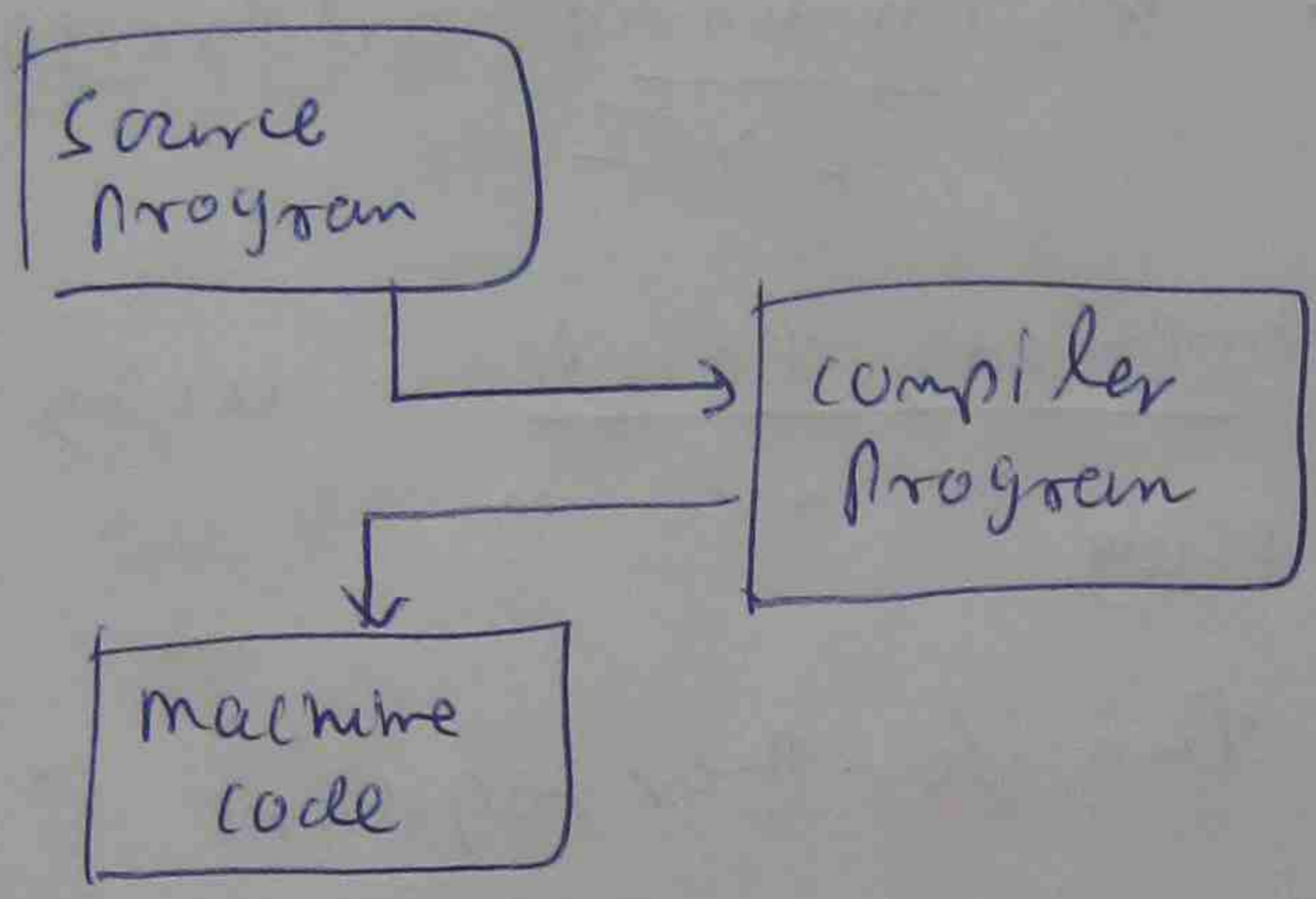
The Assembly Process

Compiler

Assembler

The compilation process

The hand assembly process



8085 Instruction set

MOV	A, A	7F
	A, B	78
	A, C	79
	A, D	7A
		↓
	M, L	75

Appendix (1)
Page 140 → 144

move Immediate

MVI	A byte	3E
	B byte	06
	m byte	36

Load Immediate

(Reg, Perm)

LXI	R dble	01
	S dble	11
	H dble	21
	SP dble	31

Load/Store A direct

- LDA addr 3A
- STA addr 32

Load/Store A indirect

LDA & B	0A	}	STA & B	02
LDA & D	1A		STA & D	12

Load/Store I/L direct

- LALD addr 2A
- SALD addr 22
- Exchange HL/DE
- XCHG EB

(27)

Ex MOV A R

(A) ← (R) requires 7, 8

Ex MOV A, FE

(A) ← FE (hex) requires 3 E, FE

operation ↑ Immediate data ↑

Ex STA 20 F2

↑
32 operation

↑ most significant byte of memory address
↑ least significant byte of memory address

pb write the hand coding for the following operation

① the program load, a value into A register using immediate addressing, at the address 2000

② then loads this value into two further registers B and C using register addressing.

3 also finally register pair HL and DE are loaded with 2020 & E027 using immediate addressing

④ then their contents are exchanged using register addressing.

(28) Assembly Instruction

Line no.	Memory		Assembly Instruction		Comment
	Address	Content	Mnemonic	Op1 Op2	

Line no	Memory		Label	ASSEMBLY			Comment - end
	ADDRESS	CONTENT		Mnemonic	Op1	Op2	
2000 1	2000	3E		<u>MVI</u>	A	FE	(A) ← FE hex
	2001	FE		↑ 3E for 2000			
	2002	47		MOV	B	A	(B) ← (A)
	2003	<u>4F</u> ↑ Although MOV(B) is 48 data finally reached		MOV	C	B	(C) ← (B)
	2004	21 for LXI H		LXI	H	2027	(H) ← (C) 2027
	2005	20 ←					
	2006	80 ←					
	2007	21		LXI	D	E027	(D) ← (C) E027
	2008	27					
	2009	E0					
	200A	E3		XCHG			(D) ← (B) (B) ← (D)
	200B	10					

MIT 2010

200A ← 10

Data manipulation

Microprocessor - Represent data within a number of different form

Typical Arithmetic instruction.

Data representation

In general, numbers may be represented in unsigned binary, signed binary or binary coded decimal (BCD) form.

Unsigned binary

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0 = weights
0	0	1	0	1	0	1	= 43 (decimal)
0	1	0	0	0	1	1	= 70
1	0	1	0	0	0	0	= 161
1	1	0	0	1	1	0	= 204

Signed binary

26 =

2	26	0
2	13	1
2	6	0
2	3	1
	1	

= 11010

for 8 bit

$$\begin{array}{r}
 0 \\
 \uparrow \\
 + \quad 0011010 \\
 \hline
 \quad \quad \quad 7 \text{ bit} \\
 = +26
 \end{array}$$

100 =

2	100	0
2	50	0
2	25	1
2	12	0
2	6	0
2	3	1
	1	

= 1100100

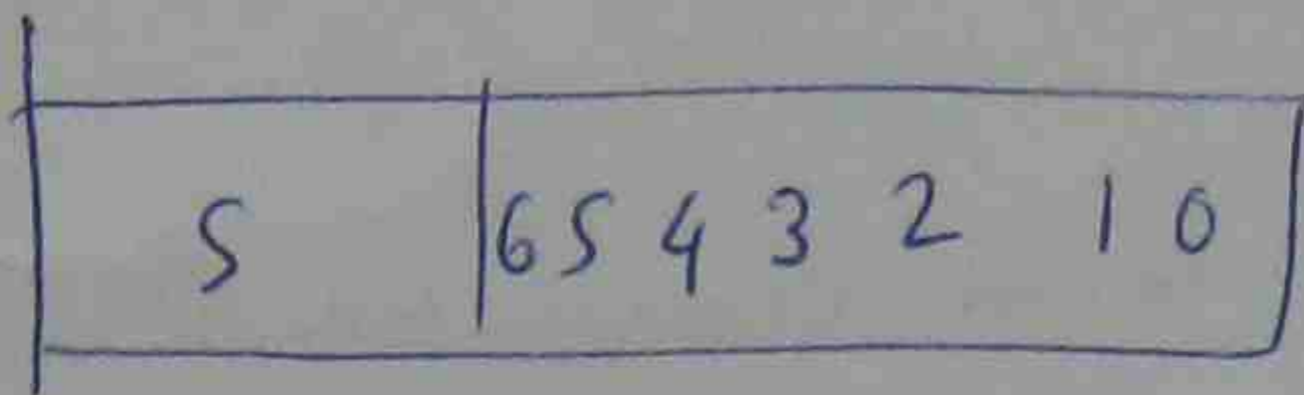
for 8 bit

$$\begin{array}{r}
 1 \\
 \uparrow \\
 1100100 = -100
 \end{array}$$

This form is not possible to perform a simple arithmetic operation

Two's complement form of representation

(39)

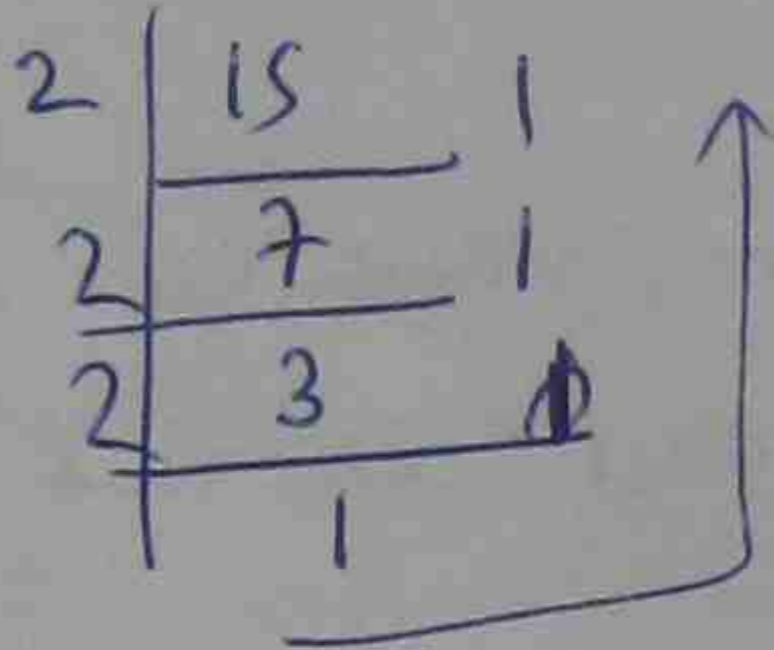


S = 0 for positive number and zero

S = 1 for negative numbers.

Ex 1

+15



1111

for 8 digit =

0000 1111

) Invert

1111 0000

+ 1

) Invert + 1

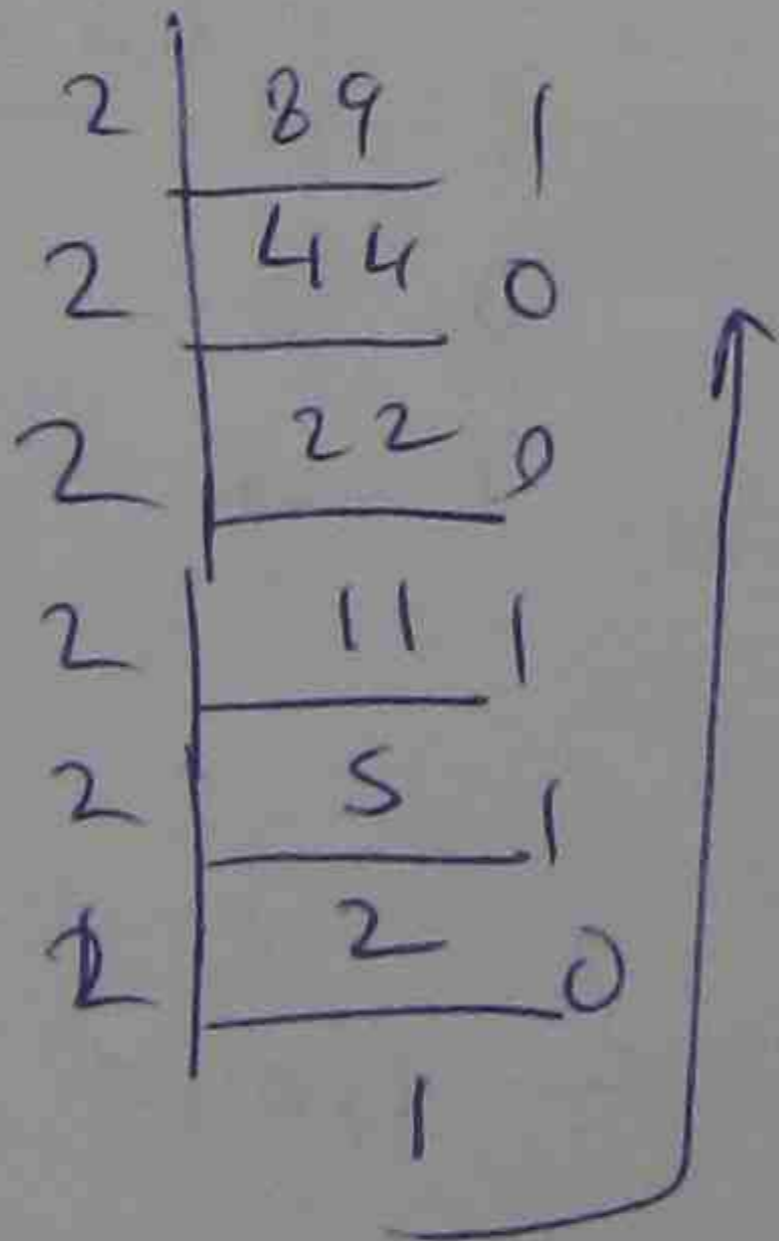
1111 0001

-15 =



Ex 2

+89 =



1011001

for 8 bit =

01011001

) Invert

10100110

) + 1

-89 =

10100111

Table Two's complement Representation

Decimal number	Two's complement
+127	0 111 1111
⋮	
+3	0 000 000 11
+2	0 000 000 10
+1	0 000 000 01
0	0 000 000 00
-1	1 111 1111
-2	1 111 1110

-3 → 11111101
 -127 10000001
 -128 10000000

Binary coded decimal (BCD)

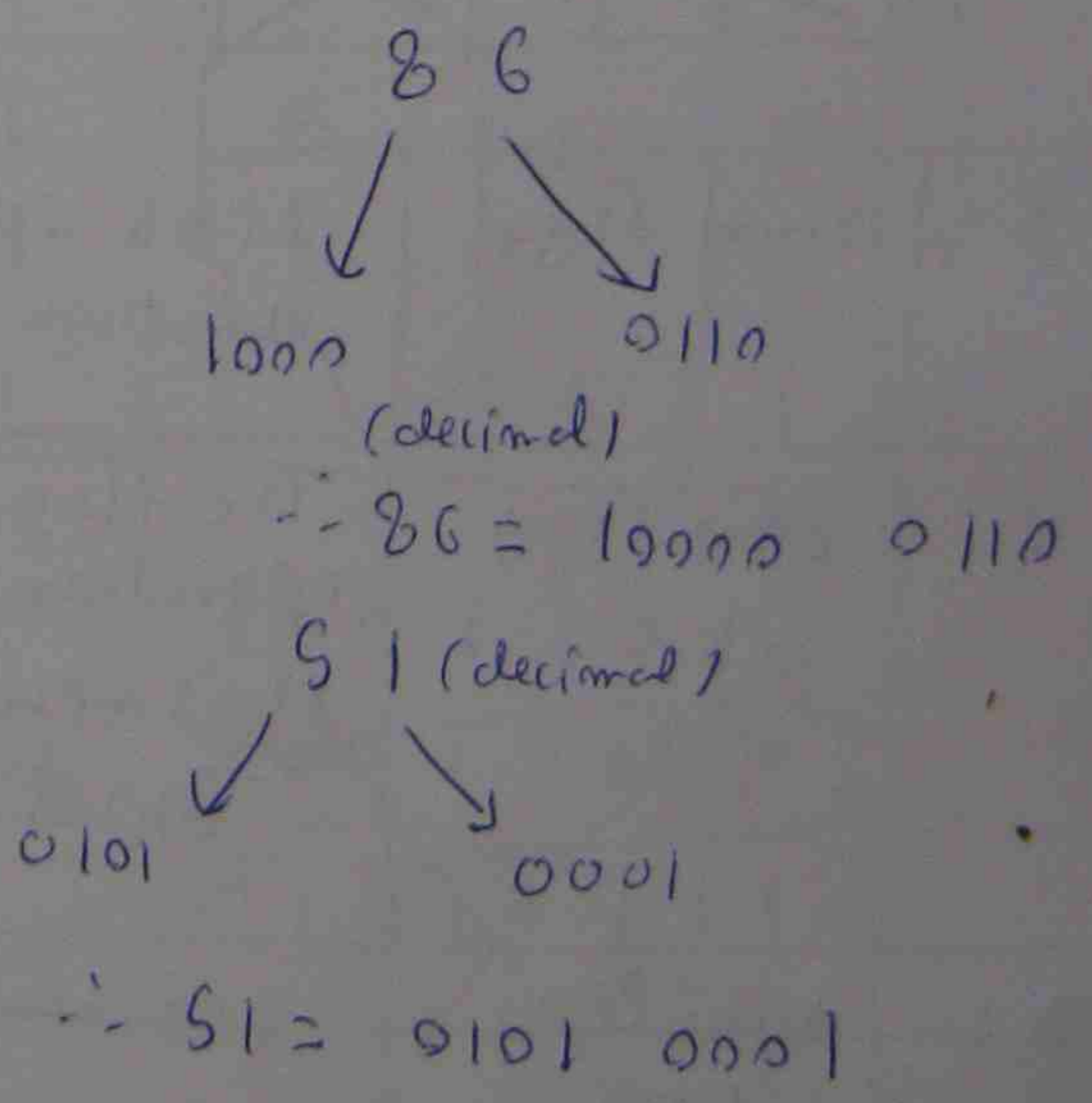
If the input data is from a decimal key pad and the subsequent output data drives a decimal display, use decimal number representation and arithmetic → provide instructions for performing arithmetic on binary coded decimal (BCD) number.

BCD representation is a subset of the hexadecimal system

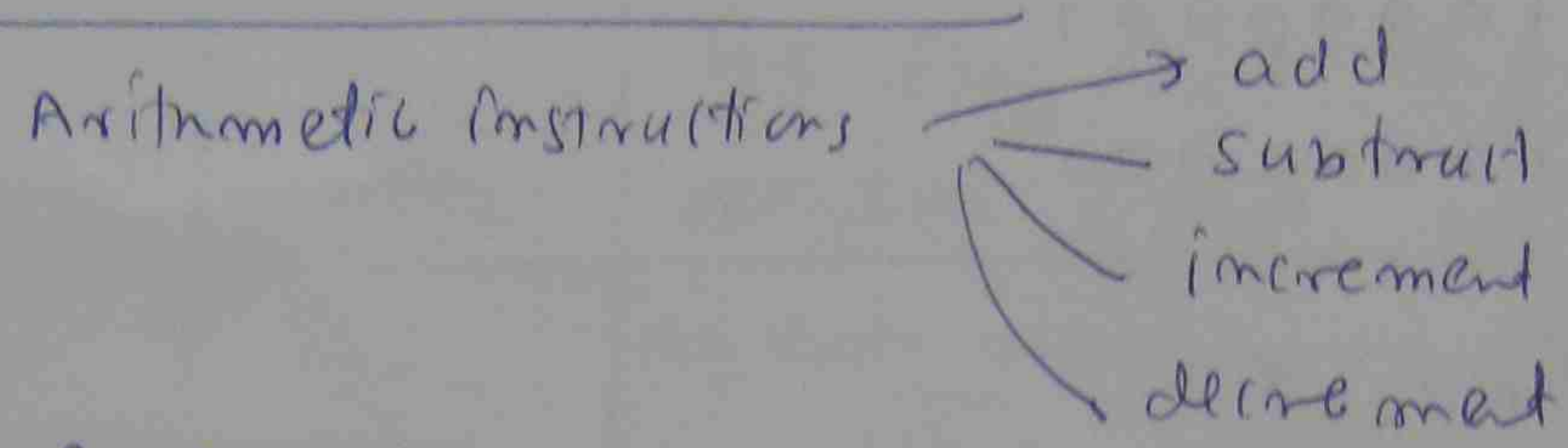
BCD code

Decimal digit	BCD code
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001

8 bit binary code may be used to store two BCD numbers.



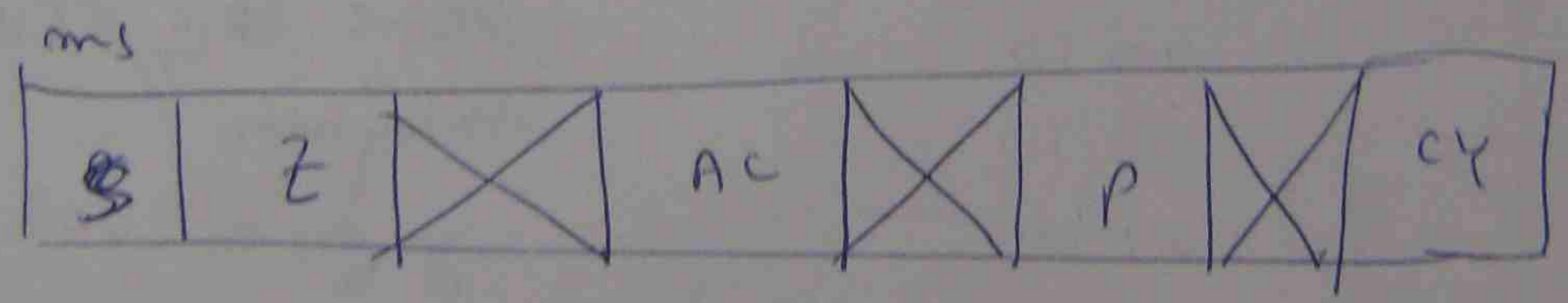
Arithmetic Instructions



8085 The instructions always involve A register and either another processor register (OR) a memory location.

— A microprocessor contains a number of different forms of these instructions so that data can be manipulated in the selected manner.

Flag - status (or) condition bit
 - which are either set (or) reset depending on the particular arithmetic instruction being carried out and the programmer is able to use and interpret these flags to manipulate data in the selected way



F register of 8085

S =	Sign flag	It is set when the result of an arithmetic operation is negative
Z	Zero flag	- The flag is set if the result of an arithmetic operation on the register is zero, otherwise it is reset. - This is used with transfer of control function
AC	Auxiliary Carry	Bcd number representation is being used It is set when the result of an arithmetic operation produces a carry out from the least significant half of the A-register.

P	Parity Flag	<p>This is used with logical operation. The flag is set if the result of a logical operation (AND, OR, XOR) produces an even number of 1's</p>
CY	Carry Flag	<p>CY is set after an ADD instruction if a carry out was generated from the A-register</p>

Addition of two bits

Bit 1	Bit 2	Sum	Carry
0	0	0	0
0	1	1	0
1	0	1	0

1st 2nd 1st 2nd

1st Bit 2nd Bit

0 + 1

A = 10011010 = 154 decimal

B = 01010111 = 87 decimal

241 decimal

100 ← carry in

001 ← sum

Addition of two bits & a carry in

Bit 1	Bit 2	Carry in	Sum	Carry out
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

10011010

+ ~~01010111~~

11001101

~~$$A = 10011010 = 154 \text{ decimal}$$~~

~~$$B = 01010111 = 87 \text{ decimal}$$~~

~~$$1001010 \text{ sum}$$~~

~~$$0010010 \text{ carry}$$~~

$$A = 10011010 = 154 \text{ decimal}$$

$$B = 01010111 = 87$$

$$11110001 = 241 \text{ decimal}$$

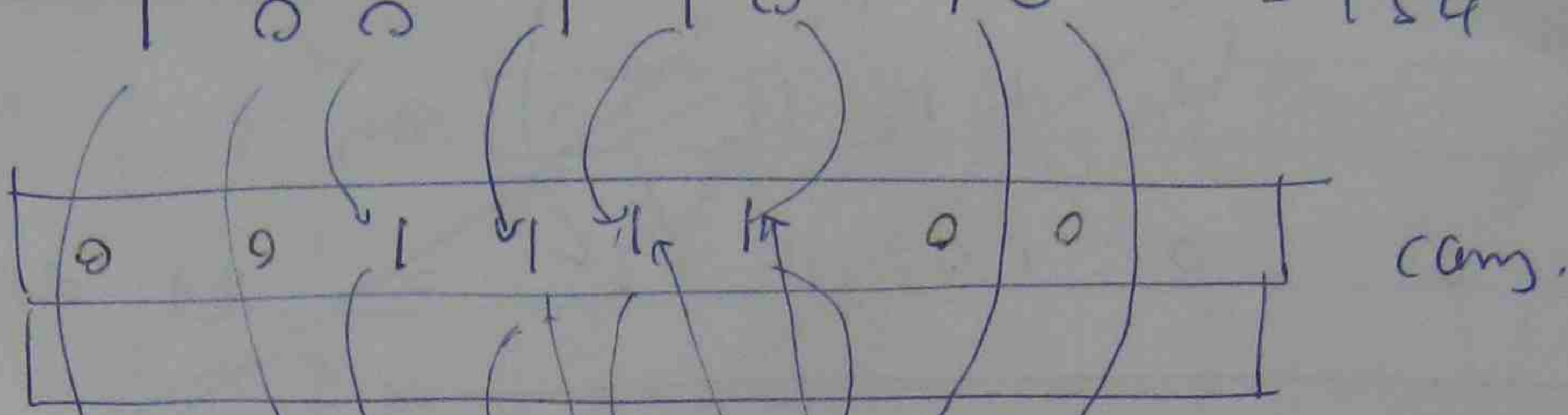
$$10011010$$

$$01010111$$

$$111000$$

34

A: 1 0 0 1 1 0 1 0 = 154 decimal

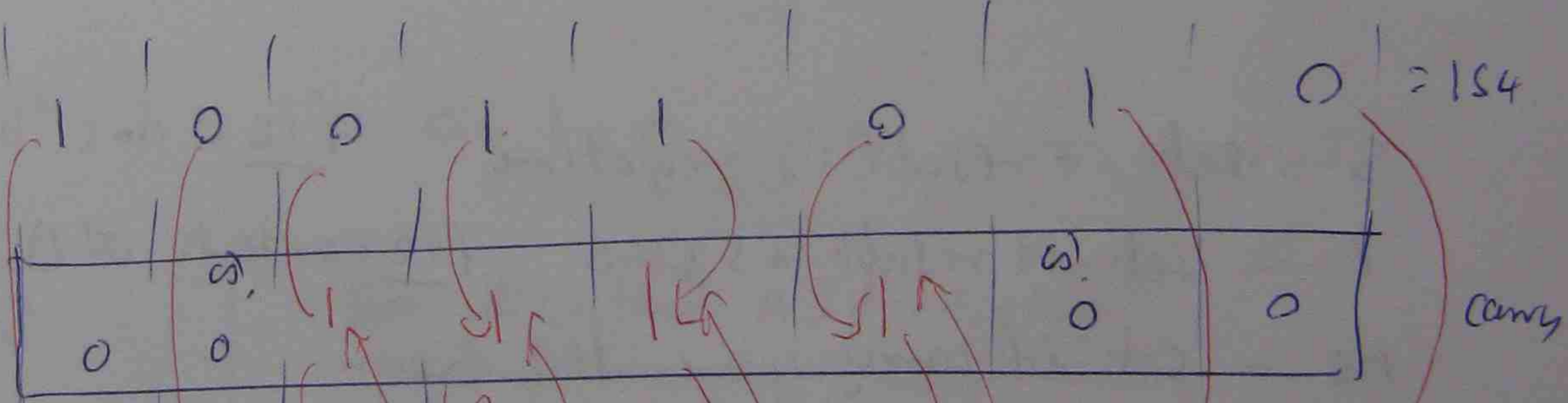


B: 0 1 0 1 0 1 1 0 = 87 decimal

1 1 1 0 0 0 1

decimal

A: 1 0 0 1 1 0 1 0 = 154

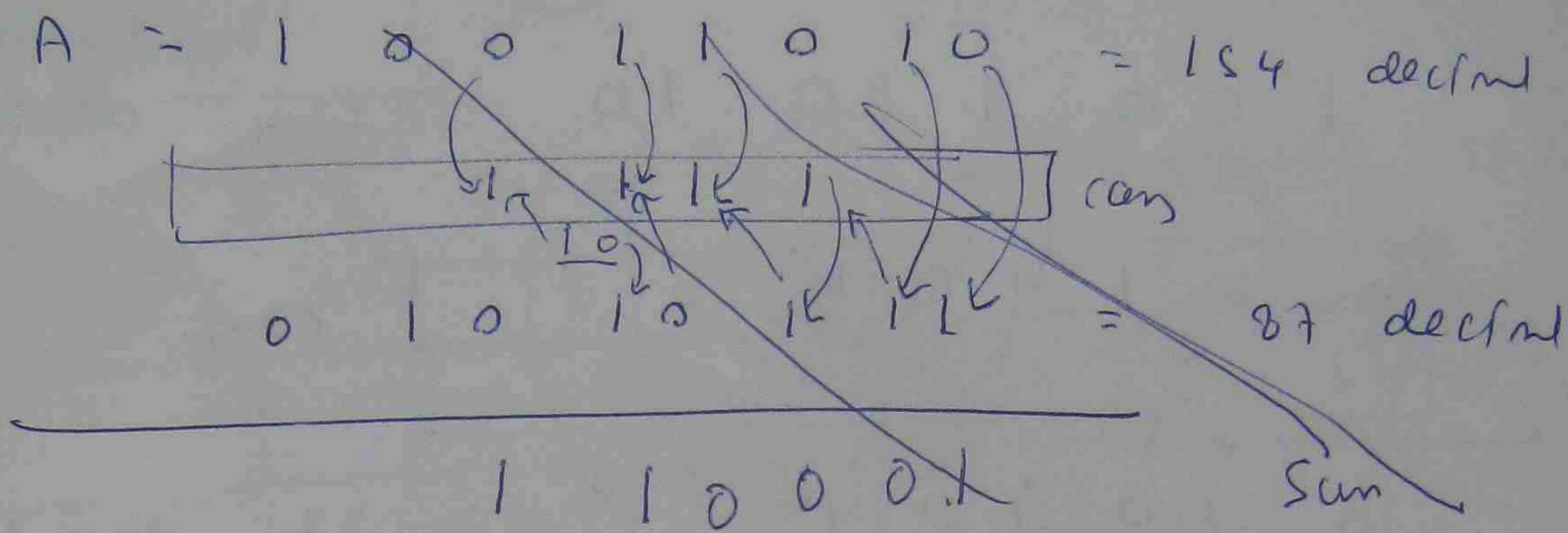


B: 0 1 0 1 0 1 1 0 = 87

1 1 1 0 0 0 1

241

decimal



Register Addressing

Ex ADD B

the contents in B register is being added to current contents of the A register

$$[A] \leftarrow [A] + [B]$$

S = set if result is negative (ie ms bit of A is 1)

Z = set if result is zero (ie contents of A are all 0s)

AC = set if carry generated from bit 3 (used with BCD arithmetic)

CY = set if carry from bit 7 (ie ms bit)

R = Reset

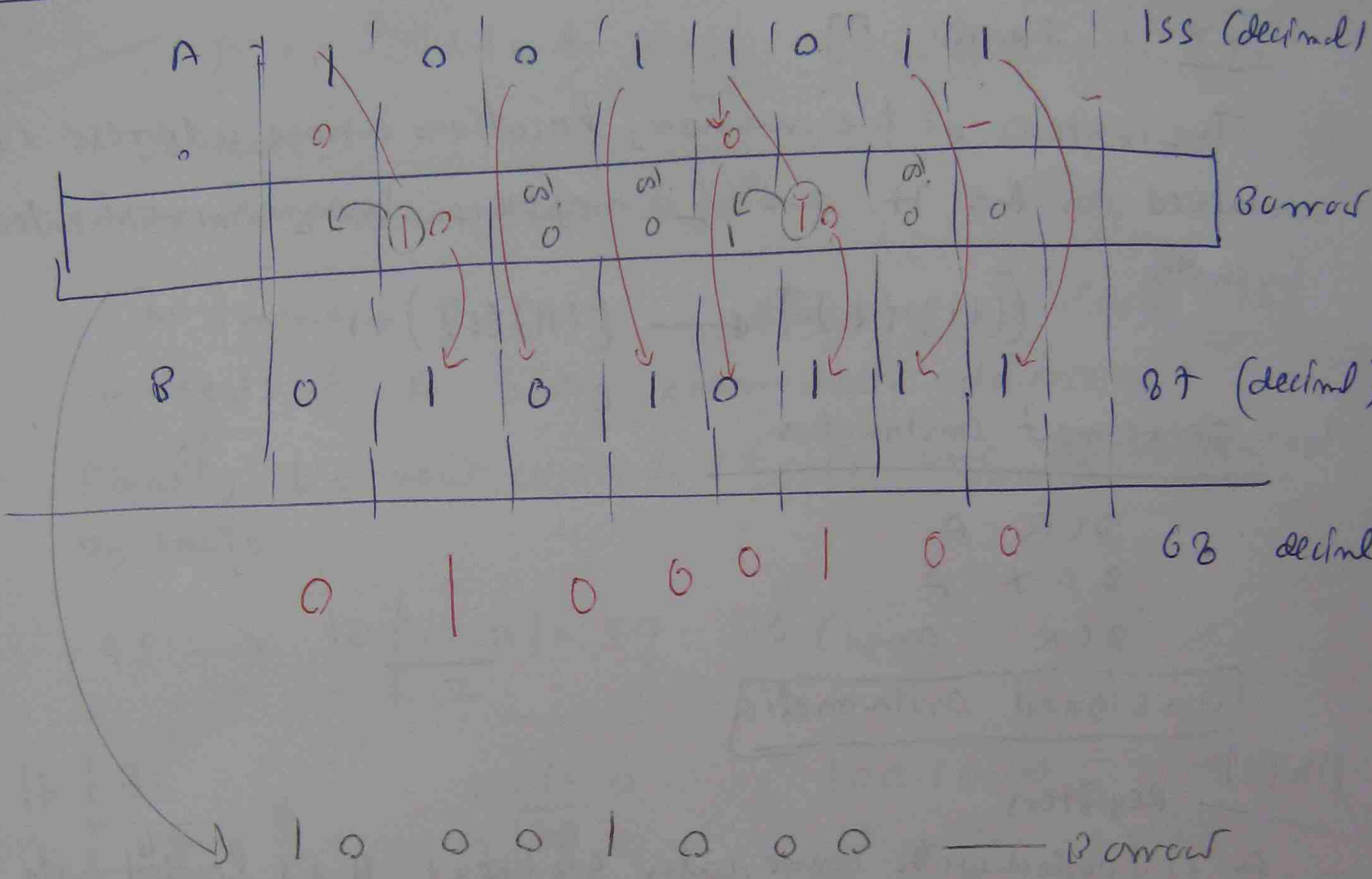
Ex ADI OF

The immediate data OF (hex) being added to the current contents of A register

$$[A] \leftarrow [A] + OF$$

all flags are affected.

Subtract



Subtract

- SUB B
- SUI OF
- SUB m

Register Addressing

EX INR A

The contents of the A-register being incremented by units.

$$(A) \leftarrow (A) + 1$$

EX INX H

combined contents of register pair H & L being incremented by units

$$(H)(L) \leftarrow (H)(L) + 1$$

Register Indirect Addressing

Ex INR m

The contents of the memory location whose address is contained in the H and L registers being incremented by unity.

$$((H)(L)) \leftarrow ((H)(L)) + 1$$

Decrement Instruction

DCR A

DCX H

DCR m

Unsigned arithmetic

Prob (1) Registers

- A is loaded with immediate data 53 (hex), B is loaded with immediate data 3A.
- Their contents are added together.
- A third numeric is then subtracted from the contents of A using immediate addressing.
- Finally the new contents of A are decremented by unity.

Assembly Instruction	Action
MUI A, 53	$(A) \leftarrow 53(\text{hex})$
MUI B, 3A	$(B) \leftarrow 3A(\text{hex})$
ADD A	$(A) \leftarrow (A) + (B)$
SBI 8C	$(A) \leftarrow (A) - 8C(\text{hex})$
DCR A	$(A) \leftarrow (A) - 1$

DCR A

$(A) \leftarrow (A) - 1$

16 bits arithmetic

8 bit - only B

16 bit B & C → BC

8 bit only D

16 bit only D & E → DE

8 bit - only H

16 bit - H & L → HL

EX JNB B

(B)(C) ← (B)(C) + 1

EX DCX D

(D)(E) ← (D)(E) - 1

EX DDB B

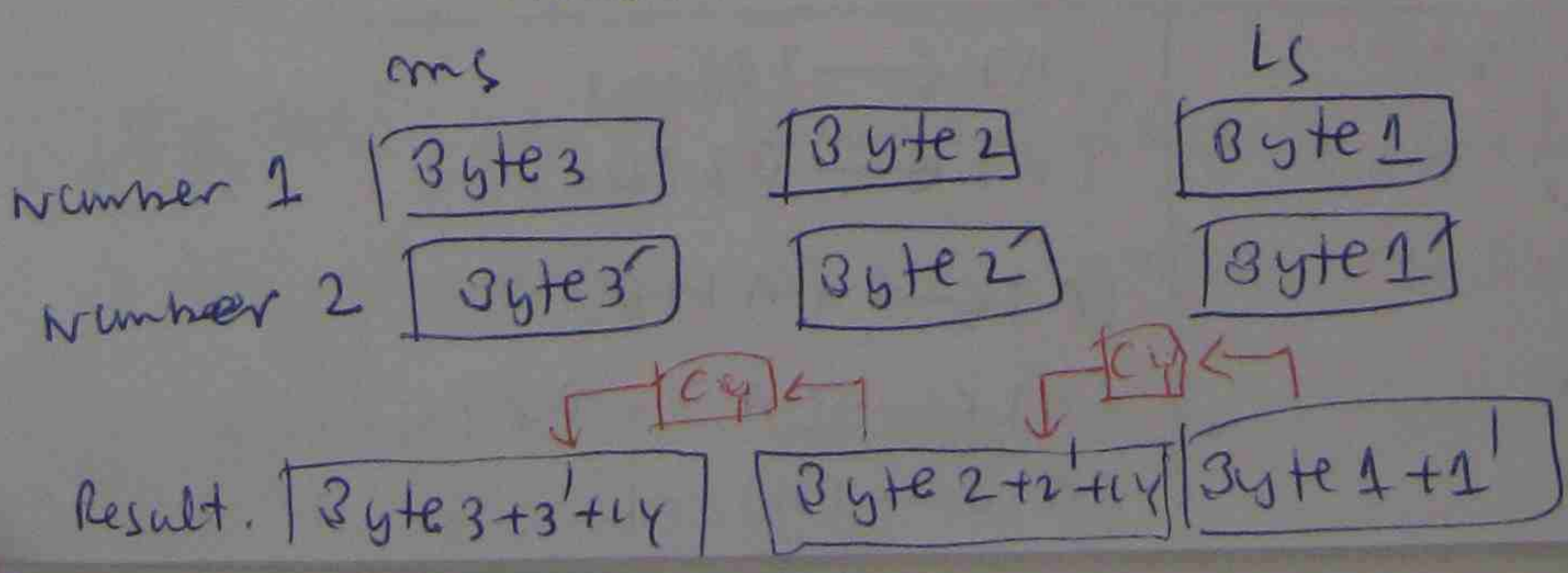
16 bit contents of the register pair B,C being added to the 16 bit contents of the register pair H,L.

(H)(L) ← (H)(L) + (B)(C)

multiprecision Arithmetic - The carry flag

If more than 16 bit accuracy is required, there are no single instructions available for performing arithmetic operation and instead, a number of instructions must be used.

Addition of two 24 bits



(40)

Ph 2 EY ADC M

- (1) The contents of the memory location whose address is contained in registers H & L
- and
- (2) The contents of CY Flag being added to the contents of A Register
- (3) The result is placed in the A register and all flags are affected.

$$(A) \leftarrow (A) + ((H)(L)) + (CY)$$

EY SBB M

- (1) The contents of the memory location whose address is contained in registers H & L
- and
- (2) The contents of CY Flag being subtracted from the contents of A register
- (3) The result is placed in the A register and all flags are affected.

$$(A) \leftarrow (A) - ((H)(L)) - (CY)$$

Ph 3 multiprecision Arithmetic

24 bits (3 byte) number. first is loaded in 2020, 2nd is loaded in 2021, 3rd is loaded in 2023.

< 24 bits (3 byte) number which is stored in the three consecutive memory location starting at address

2080 to 2082)

They are added together at 2083.

24 bit result replaces first number.

1st → LX I H 2083 (Initialize to contain 2083)

1

Load the number at 2080	LDA 2080	} Add 1st pair of byte
Add memory	ADD M	
Stored	STA 2080	

INX H Increment H/L

2

Load the number at 2081	LDA 2081	} Add 2nd pair of bytes together with carry
Add memory	ADD M	
continue to add memory	STA 2081	
Stored		

INX H Increment H/L

3

Load the number at 2082	LDA 2082	} Add 3rd pair of bytes together with carry
Add memory & carry	ADC M	
Stored	STA 2082	



Replace

Add, the result is added to 2083 which replace the first number

Program

```

LXI H, 2083
LDA 2080
ADD M
STA 2080

JNZ H

LDA 2081
ADC M
STA 2081

JNZ H

LDA 2082
ADC M
STA 2082

```

BCD Arithmetic

Prob 1 Add 62 BCD and 25 BCD

$$62 = \begin{array}{cc} 6 & 2 \\ \downarrow & \downarrow \\ 0110 & 0010 \end{array}$$

$$25 = \begin{array}{cc} 2 & 5 \\ 0010 & 0101 \end{array}$$

$$\therefore 62 + 25 = \begin{array}{cc} 0110 & 0010 \\ + & 0010 \\ \hline 1000 & 0111 \end{array}$$

No carry beyond 4 bits
CY = 0

$$\begin{array}{cc} 1000 & 0111 \\ \downarrow & \downarrow \\ 8 & 7 \end{array} \text{ BCD}$$

Prob 2 Add 79 BCD & 16 BCD

$$79 = \begin{array}{cc} 7 & 9 \\ \downarrow & \downarrow \\ 0111 & 1001 \end{array}$$

$$79 + 16 = \begin{array}{cc} 0111 & 1001 \\ + & 0001 \\ \hline 1000 & 1111 \end{array}$$

$$\begin{array}{cc} 1000 & 1111 \end{array}$$

ph 3 add 39 BCD & 48 BCD

39 = 3 9
 ↓ ↓
 0011 1001

48 = 4 8
 ↓ ↓
 0100 1000

39 + 48 =

0011	1001	
0100	1000	
1000		0001
		↑ carry
		1000 0001
		↑ normal BCD

Carry ∴ cy = 1
no bit to put ∴

But 39 + 48 = 87 ∴ reqd. BCD = 87
 ↓ ↓
 1000 0111

Required BCD 1000 0111 > Normal BCD 1000, 0001

∴ AC = 1
 ∴ cy = 1, AC = 1

To correct it 1000 0001 will need to be added with 0110

∴

1000, 0001	
+ 0110 (6)	
1000 0111	

to get 1000, 0111 or 87 hex

(+6) hex is corrected BCD sum (0110)

∴ Decimal Adjust Accumulator DAA is utilized.

< AC Flag is set 6 is added to A register >

Ph Add the following BCD numbers & make correction.

(a) 47 BCD + 32 BCD ~~(b) 81 BCD + 69 BCD~~

(a)
 47 BCD = 0100 0111
 32 BCD = 0011 0010

79 BCD 0111 1001 ← normal BCD
 ↓ 1001
 0001, ~~0001~~ +
 0000 0000 ← corrected BCD
 ↑
 Required BCD

The same

~~(b) 81 BCD =~~

Ph Subtract the following BCD numbers and make the correction

(a) 47 BCD - 32 BCD (b) 81 BCD - 69 BCD

(a)
 47 BCD = 0100 0111
 - 32 BCD = 0011 0010

 15 BCD = 0001 0101

Borrow 10

~~81 BCD~~
 - 69 BCD

 12 BCD

Reqd BCD = 0001, 0101 ← The same
 (b) ~~81 BCD~~ = ~~1000~~, ~~0001~~
 - 69 BCD = 0110, 1001

 12 BCD = 0010 1000 ← normal BCD difference

∴ Normal BCD ≠ Reqd. BCD difference ∴ AC > 1

∴ Normal BCD = Reqd. BCD difference ∴ AC = 0

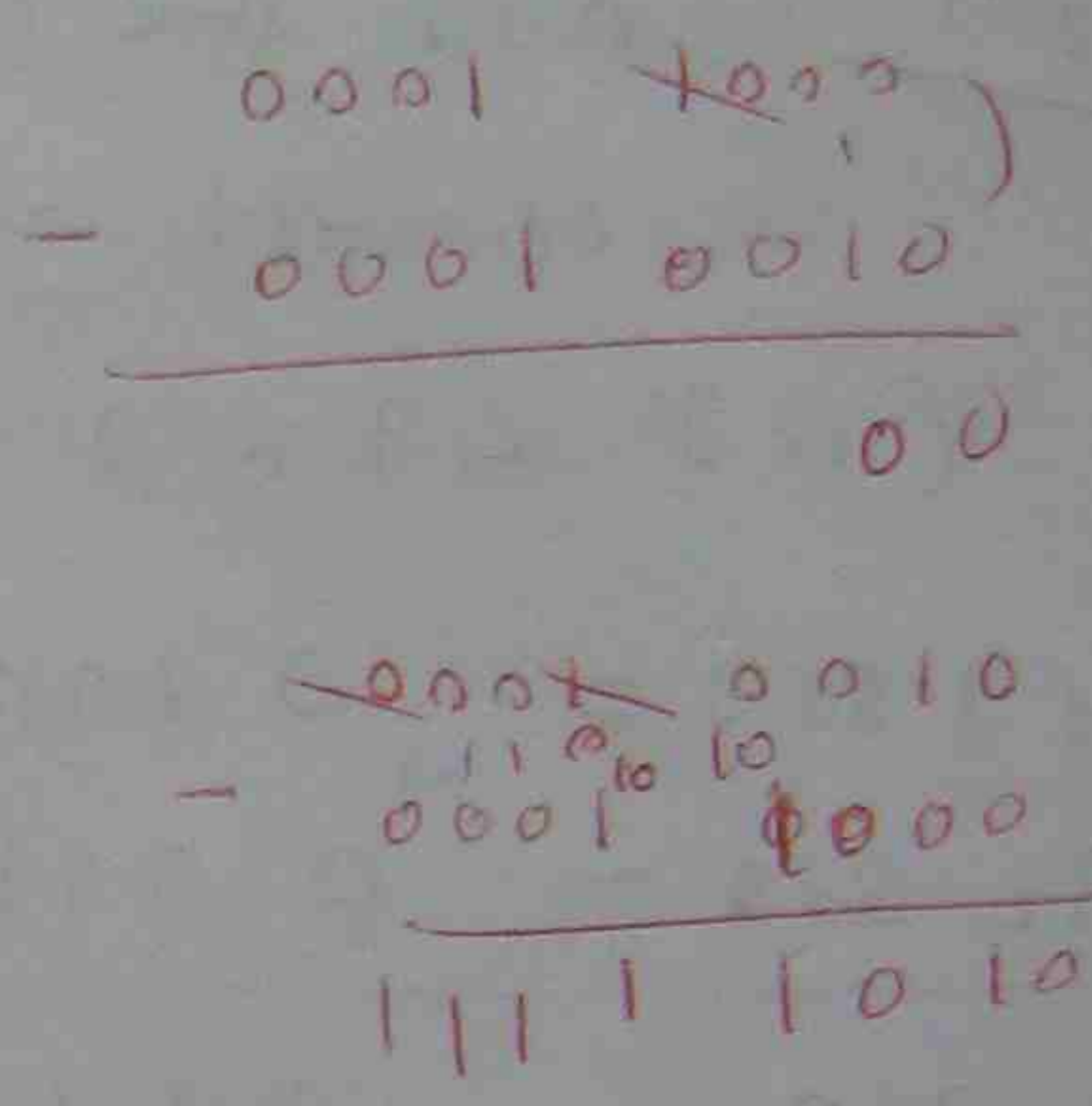
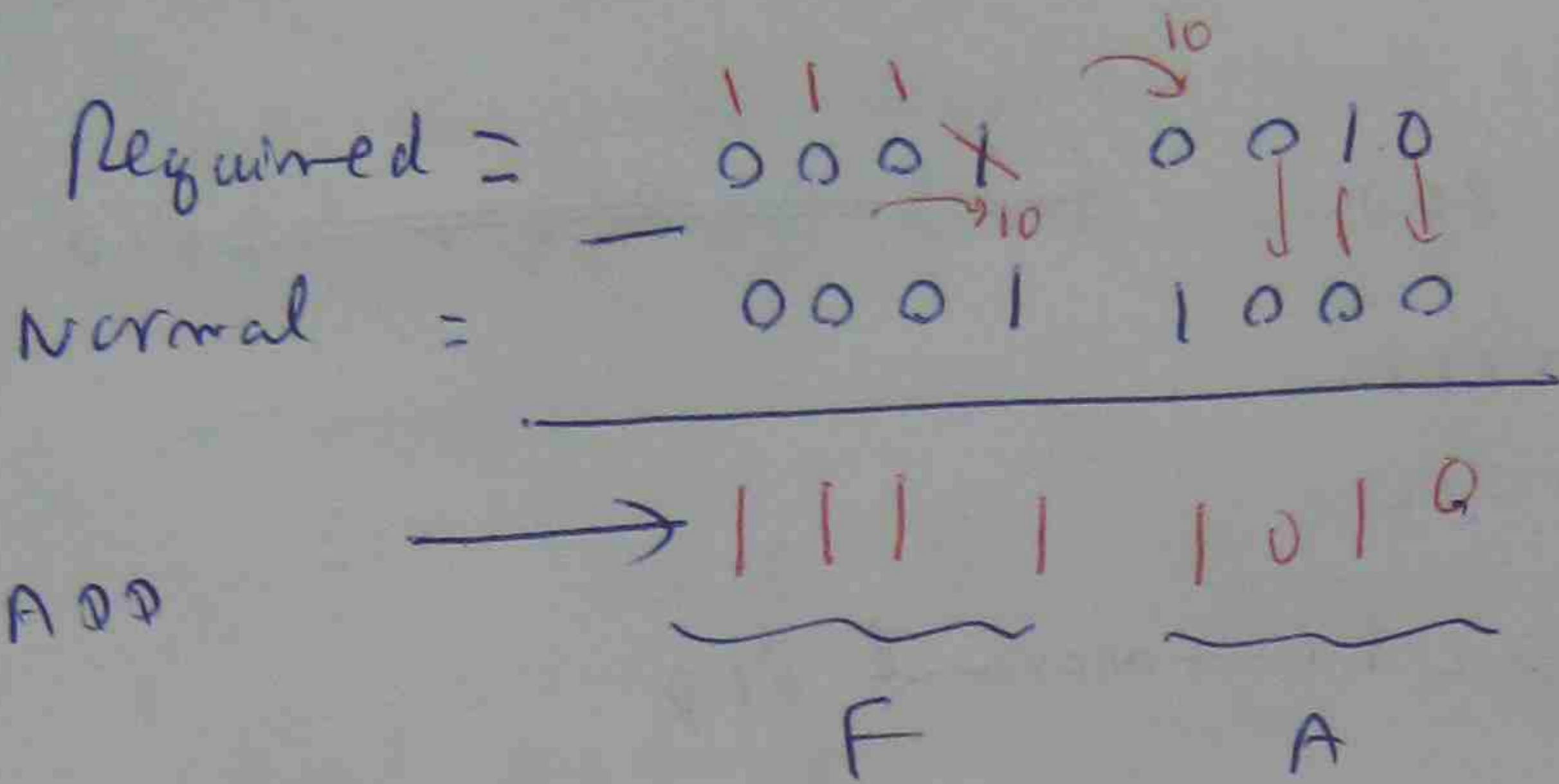
∴ Normal BCD = Reqd. BCD difference ∴ AC = 1

∴ Normal BCD ≠ Reqd. BCD difference ∴ AC > 1

∴ Normal BCD = Reqd. BCD difference ∴ AC = 0

∴ Normal BCD = Reqd. BCD difference ∴ AC = 1

Correction?



Hex Decimal numbers

4 bit binary

Hex decimal symbol

0000	—————	0
0001	—————	1
0010	—————	2
0011	—————	3
0100	—————	4
0101	—————	5
0110	—————	6
0111	—————	7
1000	—————	8
1001	—————	9
1010	—————	A
1011	—————	B
1100	—————	C
1101	—————	D
1110	—————	E
1111	—————	F

Logical operation

First number

2nd number

EX A is loaded at 2020 and B is loaded at 2021

They are added together and correction is made
The result is put into 2022 and 2022 replaces first number

Replace

LXI H 2022

1st

Load the number at 2020

LDA 2020

ADD M

Correction → DAA ←

STA 2020

INX H

2nd

Load number at 2021

LDA 2021

ADC 2021

DAA

STA 2021

↓
Add

Program

LXI H 2022

LDA 2020

ADD M

DAA

INX H

LDA 2021

ADC 2021

DAA

STA 2021

Carry Cy	Upper hex digit bit 7-4	Aux carry Ac	Lower Hex bit 3-0	Correction
0	0-9	0	0-9	00
0	0-B	1	0-F	FA
1	7-F	0	0-9	A0
1	0-F	1	0-F	9A

Logical operations

47

Logical AND

The Logical AND instructions perform the bit by bit AND operation between the contents of the A-register and either immediate data (or) the contents of another processor register (or) a memory location.

Ex ANI 40

(A) ← (A) AND 40

Ex A = 0110 0100

40 = 0100 0000

SUM A & B 0 → 0
— A & B 1 → 1

(A) AND 40 = 0100 0000 *P is next odd.*

↑ ~~select smaller number~~

Logical OR

ORA B

The contents of A register are ORed with those in B register

(A) ← (A) OR (B)

Ex (A) = 0110 0100

(B) = 1001 0101

Either A or B (1)
Result (1)

(A) OR (B) = 1111 ~~0101~~
0101

~~select bigger~~

(A) + (B)

OR
Bit 1 Bit 2 Bit (1) (or) Bit (2)
0 0 0
1 1 1

1111
~~1111 0001~~

XOR function truth table

Bit 1	Bit 2	Bit 1	XOR	Bit 2
0	0		0	
0	1		1	
1	0		1	
1	1		0	

1st

2nd

Ex (A) ← (A) XOR (A)(L)

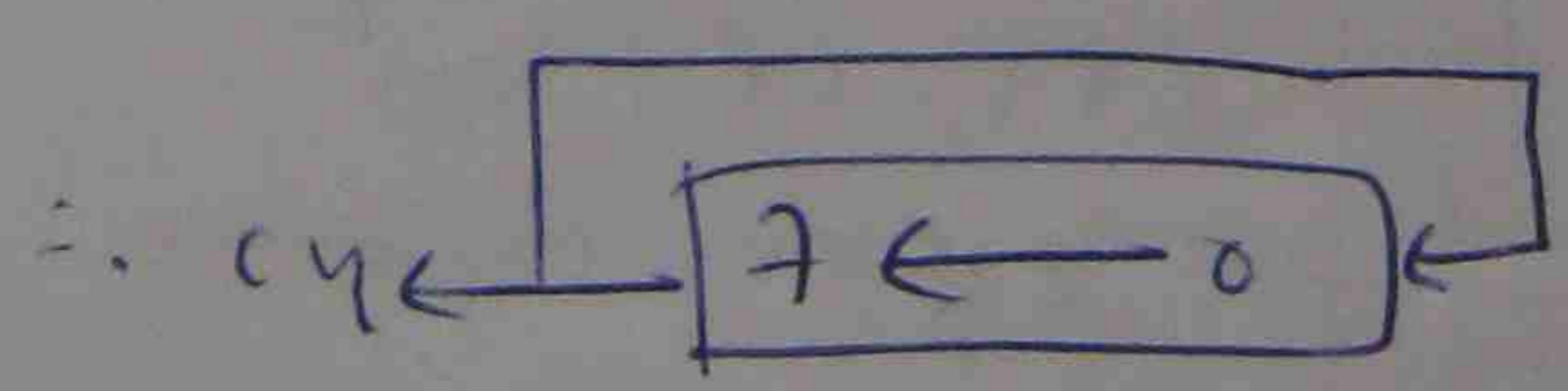
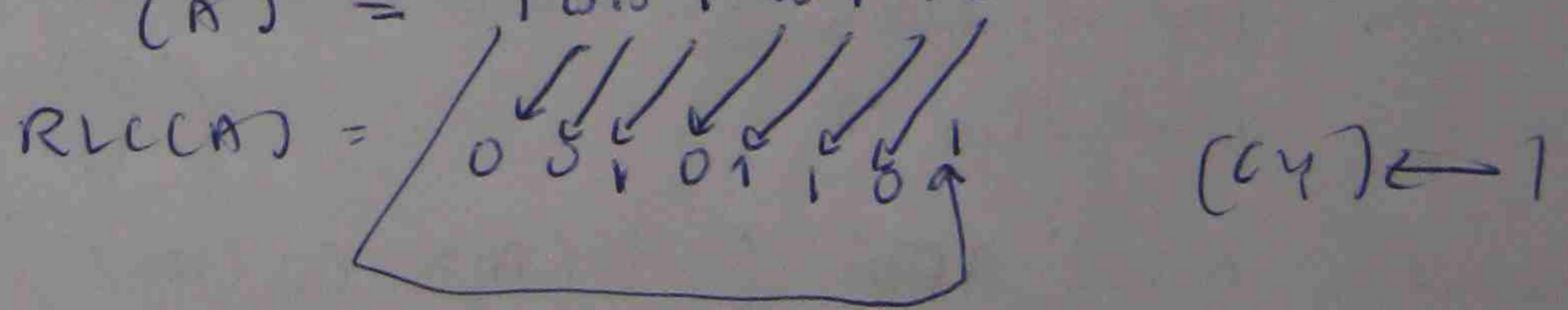
(A) = 1001 1011
 (A)(L) = 1100 1101

(A) XOR (A)(L) = 0101 0110 P is set even

Rotate

- Rotate a binary value left or right one place
- left shift = x 2 operation
- Right shift = / 2 operation
- useful for performing multiplication & division

Ex (A) = 10010110



Rotate left

Compare

Compare two values - the contents of the A-register and either immediate data (or) the contents of a processor register or memory location.

CMP B

The contents of B registers are compared with the contents of A register.

Z flag is set ~~zero~~ to 1 if the contents are equal and carry flag is set to 1 if contents of A are less than contents of B

Ex ^{The Program} First loads immediate data ^{FO & OF} into registers A and B and then perform a series of logical operation

- 1st ^{the} contents of A and B are compared
- 2nd ~~the contents~~ contents of A are then rotated left
- 3rd the new contents of A are then AND-ed with a constant 81
- 4th the resulting contents of A are OR-ed with the contents of B

3
↓
4
↓
0 2

(A) FO = 1111 0000
(B) OF = 0000 1111

Initial	MVI A, FO MVI B, OF	(A) = FO (hex) (B) = OF (hex)	
1st	CMP B	Z ← 0 CY ← 0	
2nd	RLC	(A) ← EA CY ← 1	
3rd	ANI 81	(A) ← 81, CY ← 0, P ← 1	
4th	ORA B	(A) ← 8F	

(99)

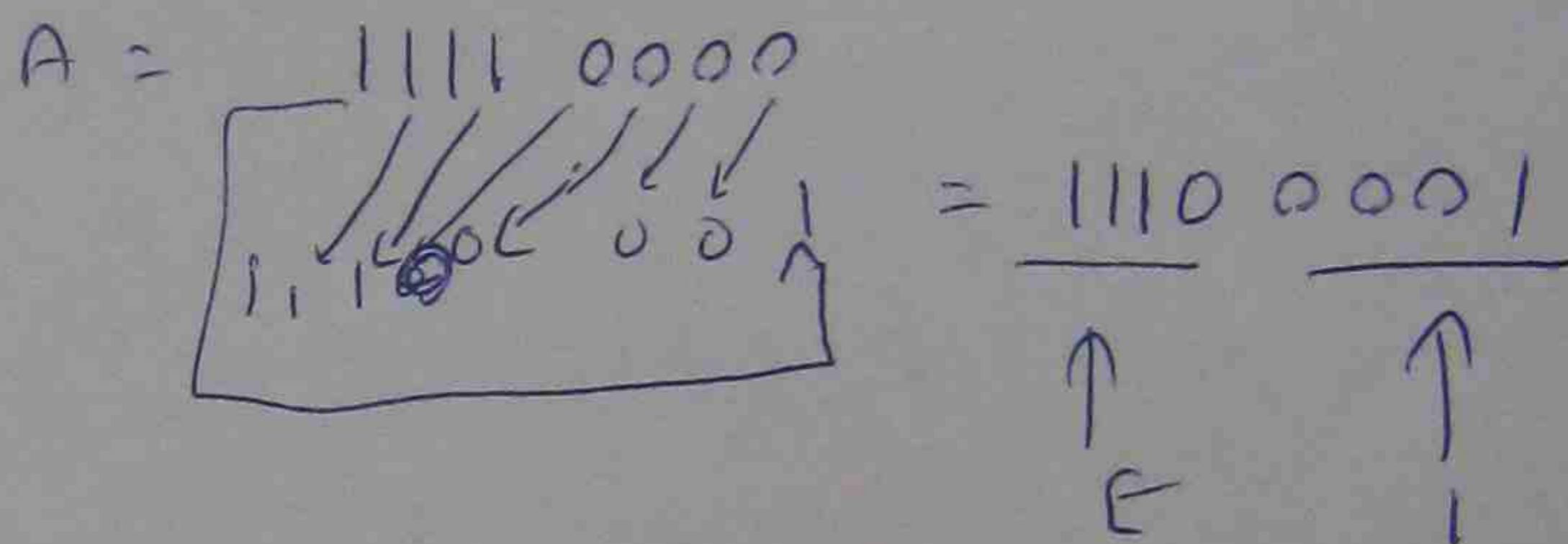
movl A fo A = 1111,0000

movl B of B = 0000,1111

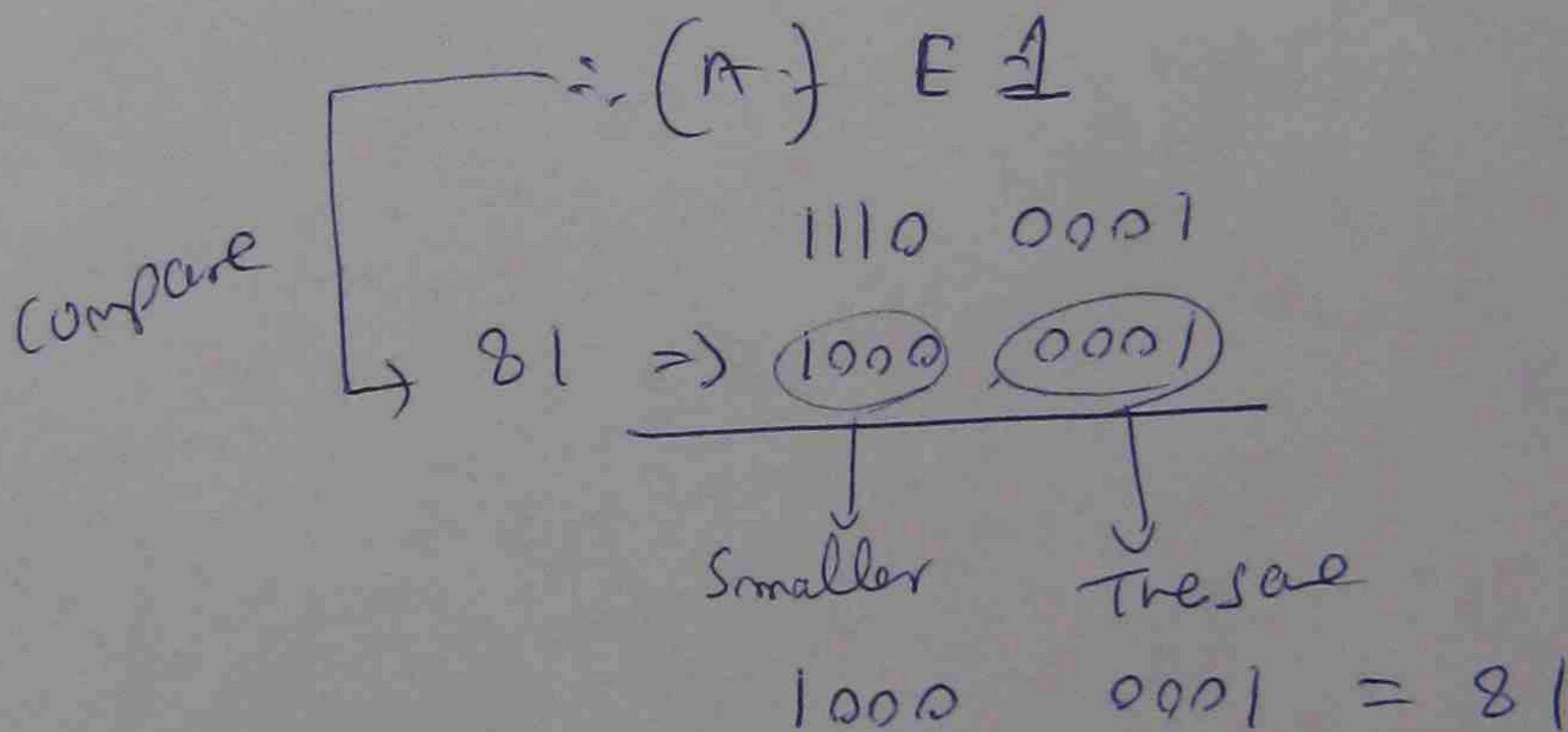
1st cmpl B A 1111,0000, B 0000,1111

∴ A > B ∴ A = 1111,0000

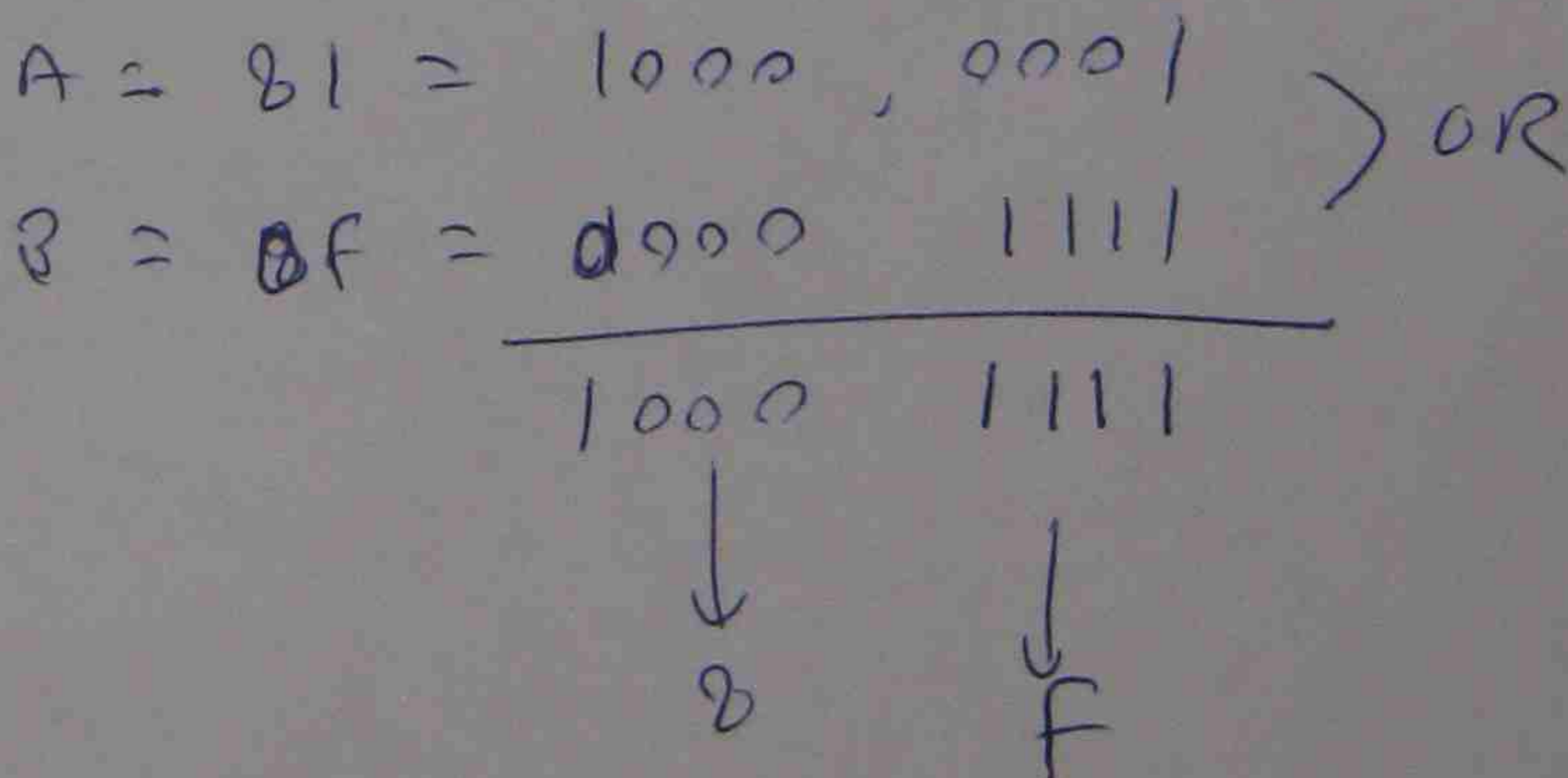
2nd RLC



3rd andl 81



4th orl B



Exercise

Look at exercise questions 4.1 to 4.8
check the answer provided for exercise 4.1 to 4.8
determine the way to approach the solution.

Take exercise with microprocessor Training software

Transfer of control

- To achieve the ability to transfer control, branch or to an instruction that is not in sequential order, it is achieved with instructions from the transfer of control group.
- All these instructions act on the program counter.
- It is possible to execute a block of instructions many times over with the number of times determined either by program data or the state of a processor flag.

Jump instructions

- Break normal sequential execution
- Branch to a different part of the program.
- Loading the address of the next out of sequence instruction in to the program counter.
- Forcing the processor to fetch the contents of this new location for its next instruction
- The new address is specified in the instruction.

 Jmp 20B3

<an conditional jump to memory location 20B3 for the next instruction>

memory address	A	C3	Jmp operation
	A+1	B3	ls byte of address
	A+2	20	ms byte of address

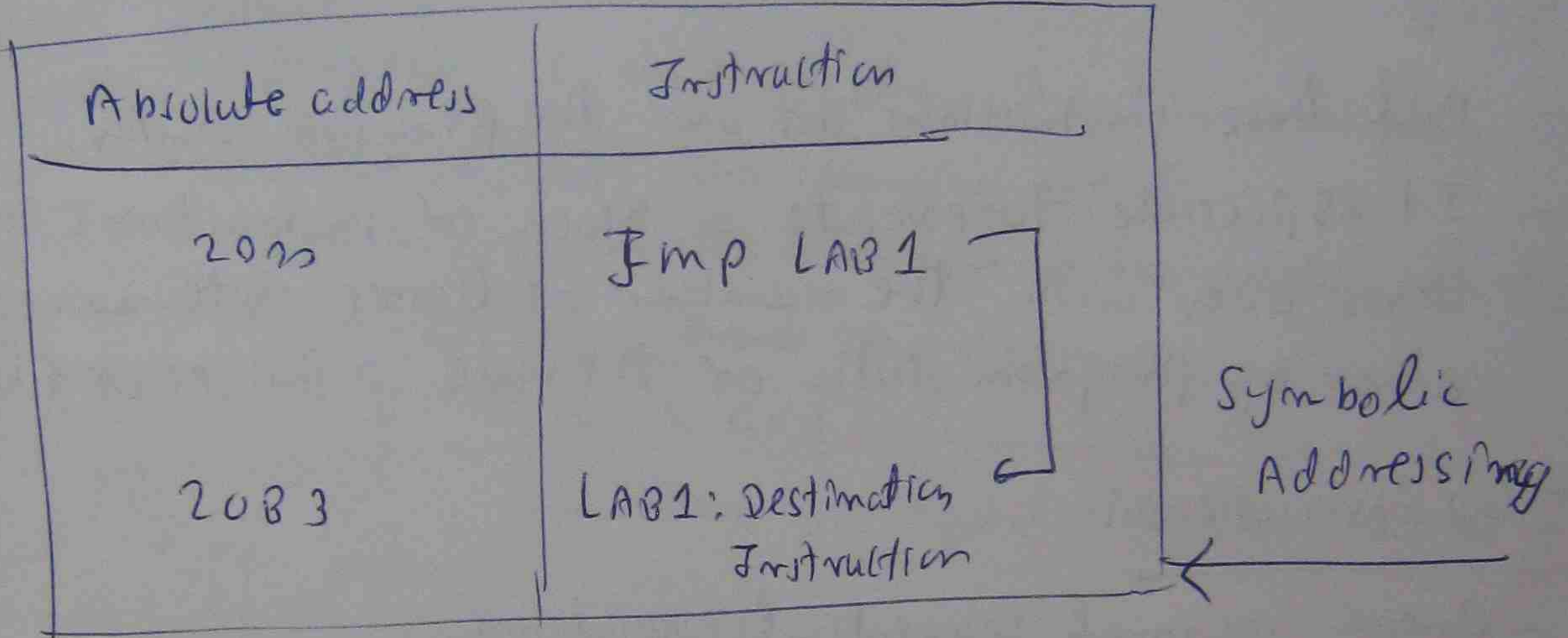
unconditional jump

Jmp LAB1



to indicate the destination address of jmp instruction

- use label to indicate jmp during assembly language development



conditional Jump

JNZ LAB1

Jump if zero flag not set to LAB1.

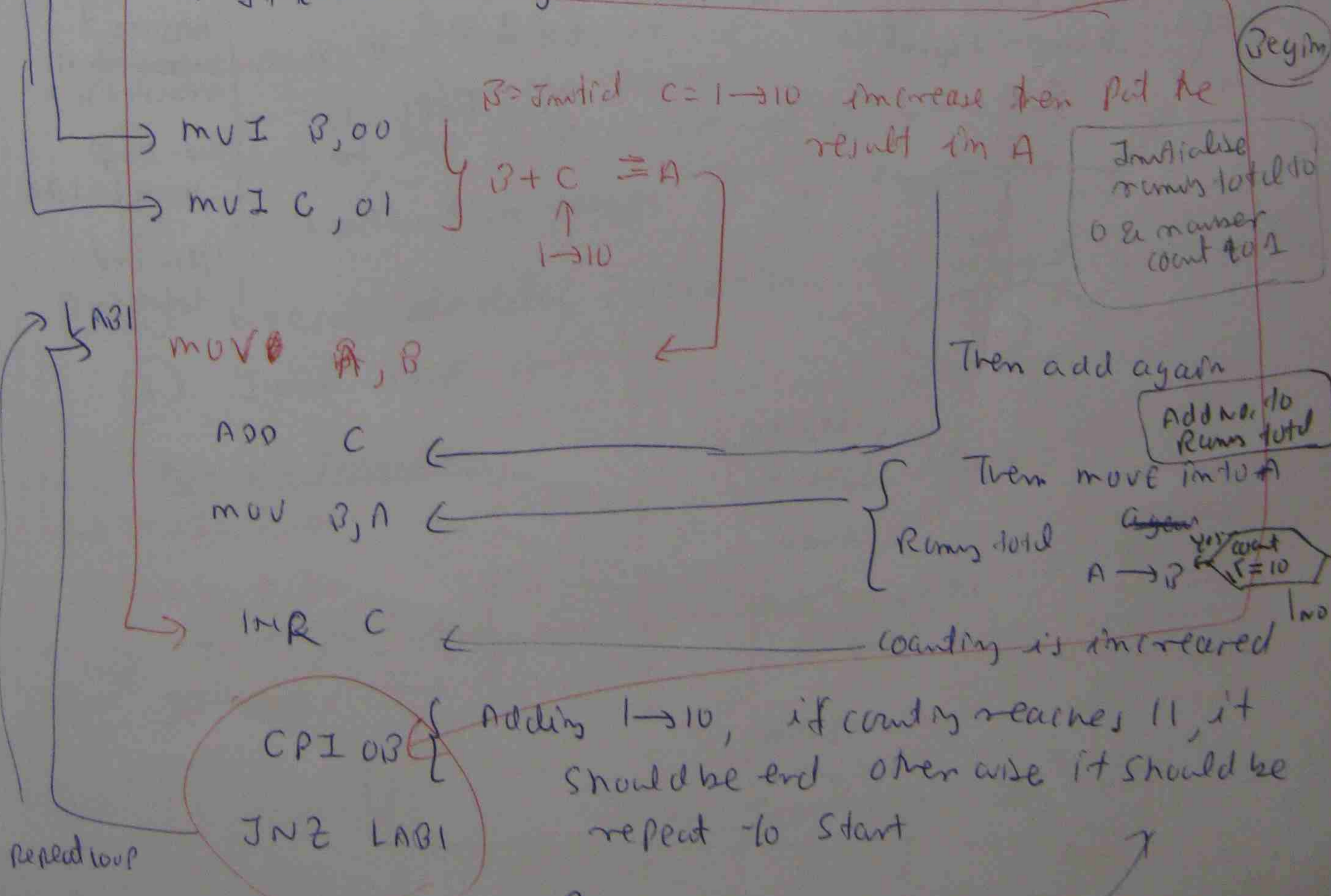
Op-code	condition	Flag status
JNZ (LAB1)	Jump if zero flag is not set to (LAB1)	Z = 0
JZ	Jump if zero flag is set to (LAB1)	Z = 1
JNC ()	Jump if carry is zero	C = 0
JC	Jump if carry is 1	C = 1
JPO	Jump if parity is odd	P = 0
JPE	Jump if parity is even	P = 1
JP JM	Jump if S = + Jump if S = -	S = 0 S = 1

Flow Chart

A diagram which indicates the sequence of, and actions required in, a program and the points where branching is required.

Prob 1 Construct the program to add together the ten numbers 1 to 10

- B register contains the running total
- C register contains the number to be added to the running total. (~~From~~ start from 1)
- Increment by 1 & add
- If count is less than or equal to 10 it will be repeated
- If the count is greater than 10 it will be ended.



```

MVI B, 00
MVI C, 01
LAB1:
MOV A, B
ADD C
MOV B, A
INR C

```

```

CPI 0B
JNZ LAB1

```

Adding 1 to 10, if counting reaches 11, it should be end otherwise it should be repeat to start

Begin

Initialise running total to 0 and number count to 1

Add number count to running total
Increment count by 1

Is ~~the~~ count less than or equal to 10

Yes

No

End

Assembly Instruction

Comment

MVI B, 00
MVI C, 01

Initialise running total
Initialise number count

LABI

MOV A, B

Bring running total to A

ADD C

Add number count

MOV B, A

Restore running total in B

INR C

Increment count

MOV A, C

Bring count to A

CPI 0B

Has count ~~reached~~ reached 11?

JNZ LABI

No - Jump back to LABI

Yes - End total in B

Time Loop

Loop

Jump Instruction

Time delay in the program

- A common requirement when a microcomputer is interfaced to other equipment is to compute a time delay in the program.
- A microcomputer-based road traffic light controller, for example, would need to compute a time delay to implement the sequencing of the light changes.
- Desired delay & loop count to be held in C-register.

Time delay NOP - NO-operation instruction.

write a program

- ph 2
- Load desired delay time parameter into C-register. *delay parameter is 02*
 - clear A
 - compare the contents of C-register with zero *if yes, end*
 - Jump ^{if} otherwise proceed. *the time is not set to zero*
 - Nom operation stages - 6 stages
 - Execute ~~all~~ delay instructions decrement C register
 - Jump loop

Assembly Instruction

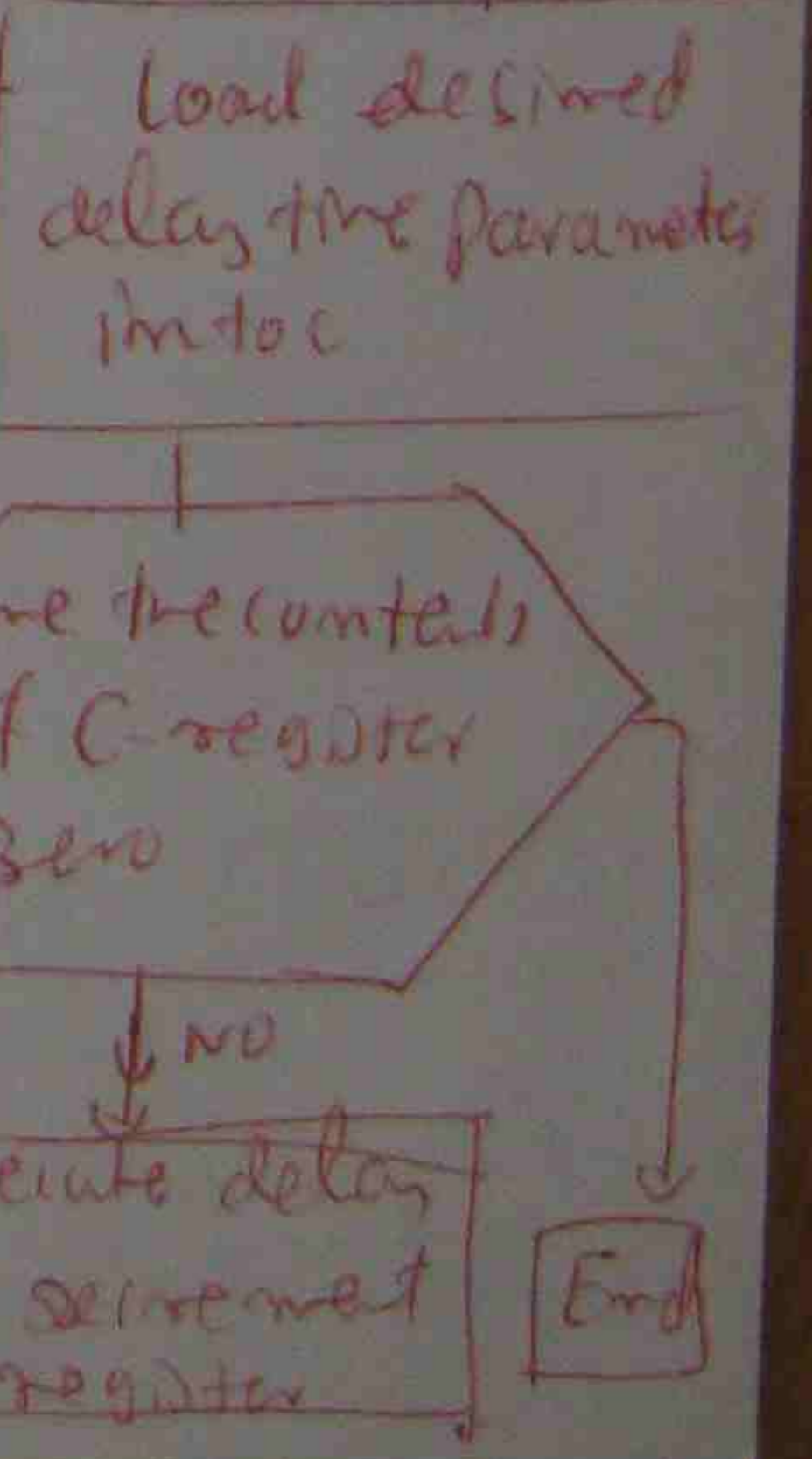
```

-> MVI C, 02
-> MVI A, 00
-> cmp 0
Loop
  JZ TIME
  NOP
  NOP
  NOP
  NOP
  NOP
  NOP
  DCR C
  JMP Loop

```

comment

set regd: delay in C
 clear A
 Are the contents of C zero?
 Nom operation instructions - provide basic time delay
 End of instruction Loop



Subroutines

Pb 2 ←

A Short Program to compute a time delay

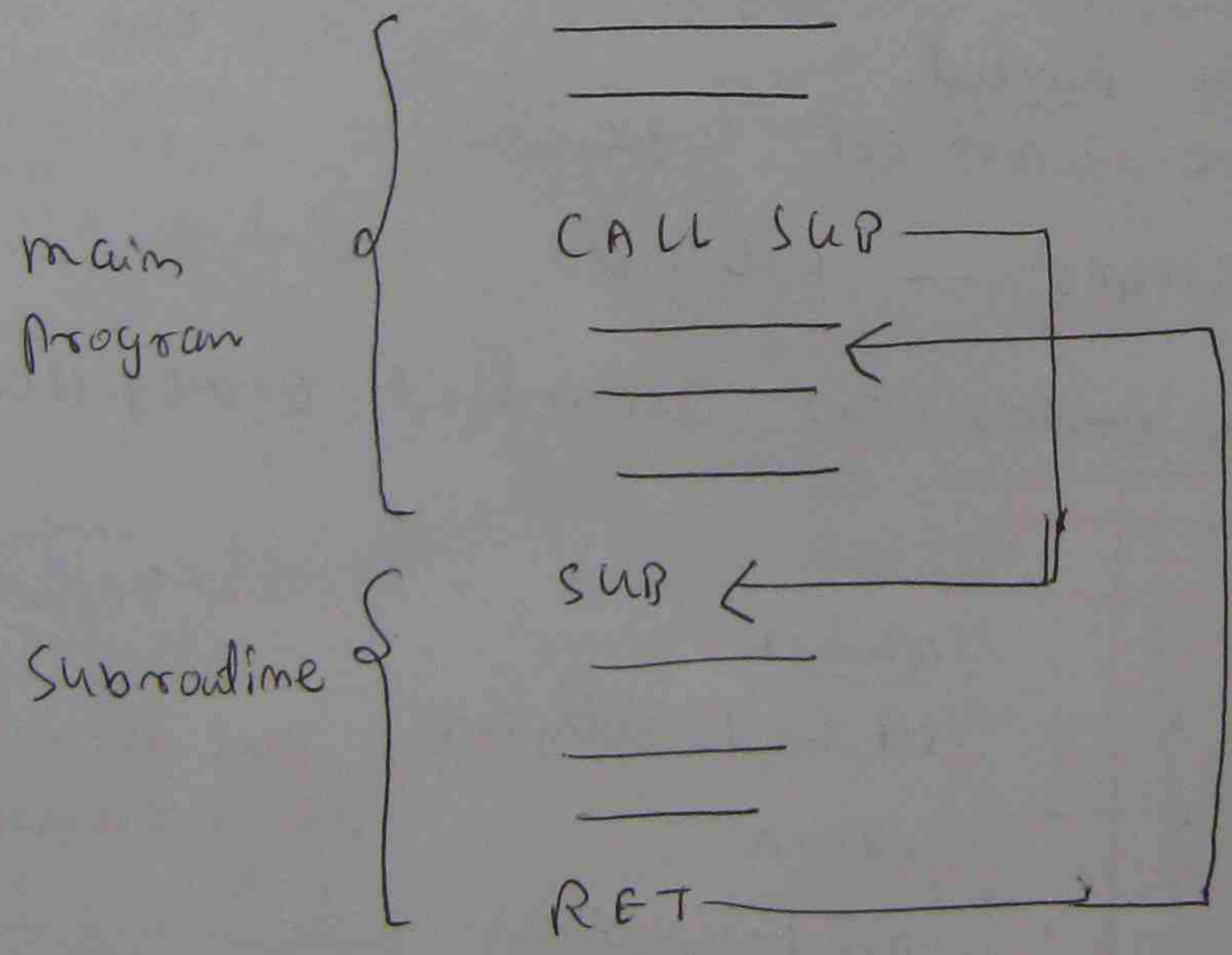
— This can be made into a subroutine by placing a RET instruction with the Label TIME at the end of the program.

Subroutine

Frequently within a program, it may be necessary to perform a particular sub-task many times over.

~~It is highly desirable~~

subroutine is designed to perform the sub task and then return control to the main instruction sequence.



Stack

A last in first out queue which is implemented as a set of successive locations either within the processor itself or more usually, in the system memory.

Stack pointer register (SP)

it points to the address which currently holds the entry at the top of the stack, and consequently its contents change as each subroutine call and return instruction is executed.

Ex ① main program memory address 2010 to 2012
 contents CD, 4B, 20 respectively

② Program counter PC - two bytes at 2013 call subroutine

③ sub routine instruction 2048 → 2049 contents XX

④ sub routine call (stack pointer) 2000 to 2002
 contents XX

⑤ Stack pointer at 2002

memory address	contents	Symbolic Instruction
2010	CD	} call sub PC=2013
2011	4B	
2012	20	
2048	XX	
2049	XX	
2000	XX	
2001	XX	
2002	XX	

→ main program }

→ subroutine }

→ stack }

SP = 2002

call instruction brought from memory
PC incremented

still blank

After subroutine call, PC=2048
SP=2000, 2002...XX

Pb 3

(58)

memory Address	contents	Symbolic Instruction
main program { 2010 2011 2012	CP 4B 20	callsup PC = 2048
Subroutine { 2048 2049	XX XX	sup
Stack { 2000 2001 2002	13 20 XX	SP = 2000

Subroutine

Pb 3

- (1) Initialise stack pointer at 2002
- (2) Load desired delay time parameter into C-register
delay parameter is 02
- (3) call subroutine which ~~is~~ is named TIMEDLY
- (4) In subroutine it consists of the following sequence:
 - (a) Clear A
 - (b) compare the contents of C if yes, goes to end
otherwise proceeds
 - (c) Jump if the time is not set to zero
 - (d) Non operation stages \rightarrow 6 stages
 - (e) Execute delay instructions. Decrement C register
 - (f) Jump loop
 - (g) Return subroutine.

ORIGINAL PROGRAM

Additional for subroutine

```

LXI SP, 2002 ← Initialise stack pointer
MVI C, 02 ← Load TIMDLY parameter
CALL TIMDLY ← call subroutine

```

```

TIMDLY ← Time delay subroutine
MOV A, 00
LOOP:  CMP A, 0
      JZ TIME
      NOP
      NOP
      NOP
      NOP
      NOP
      NOP
      DCR C
      JMP LOOP

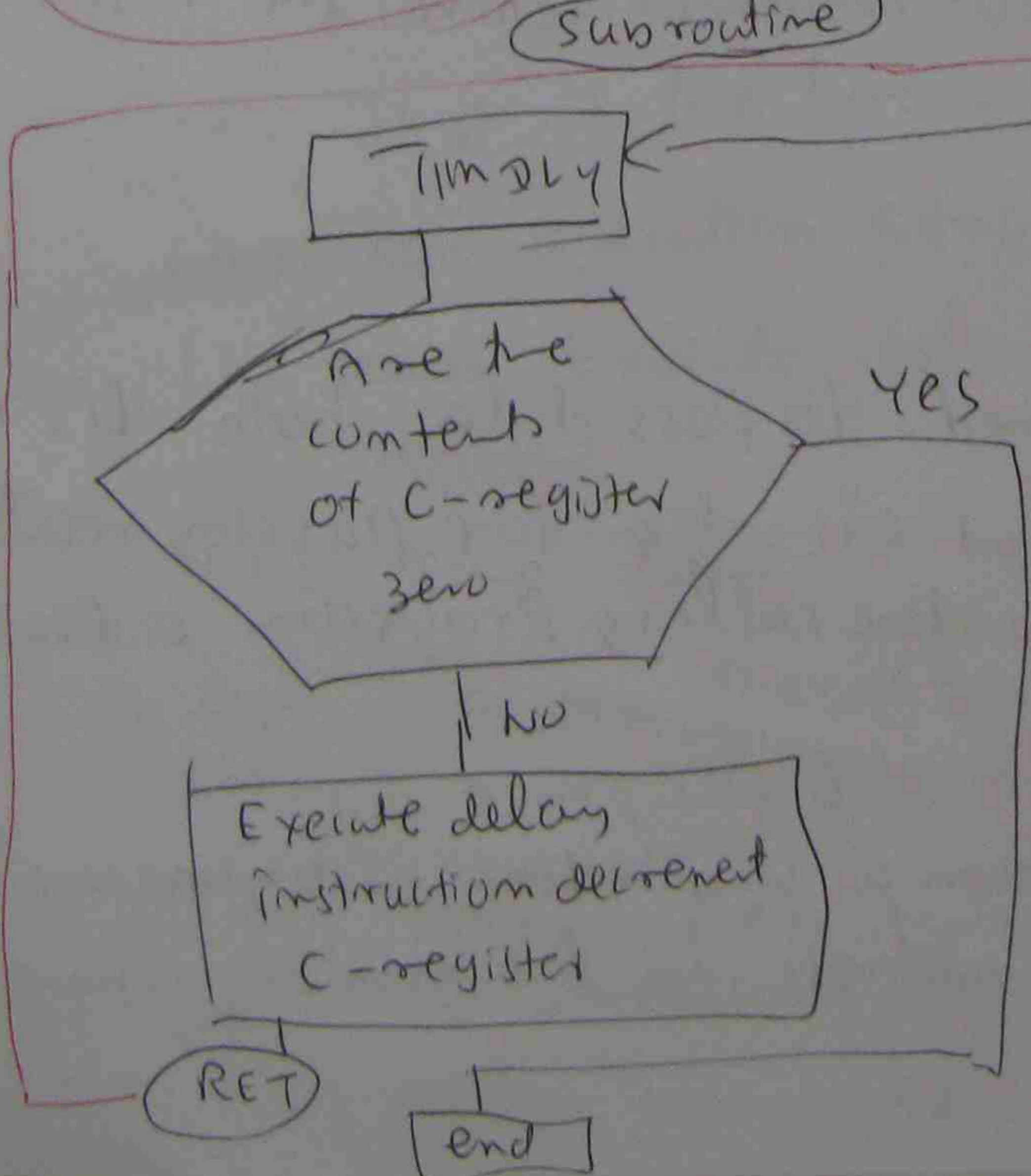
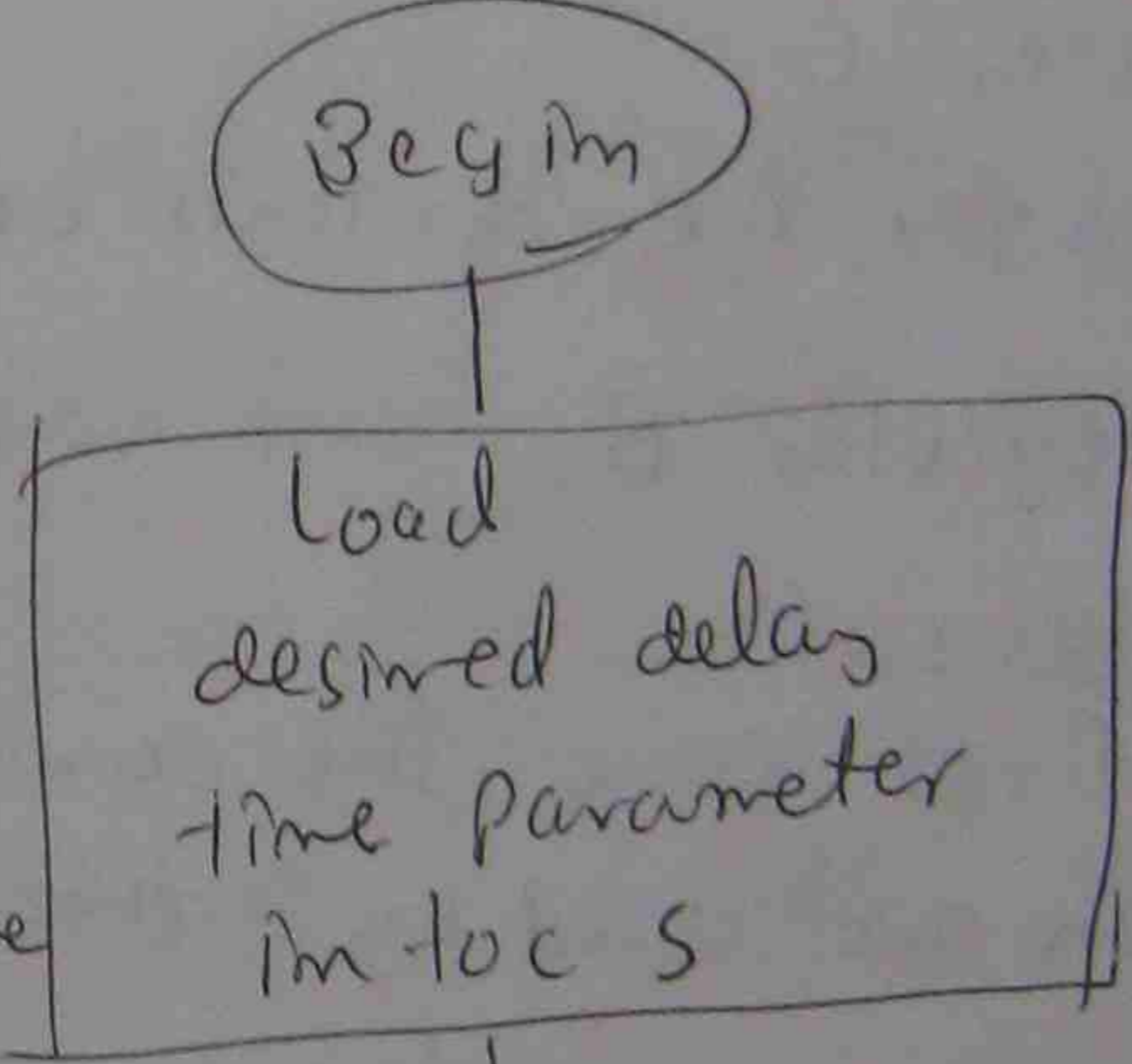
```

delay routine

Flow chart

TIME RET ← Return to sub routine

Subroutine



Stack operation

The stack may also be used as a temporary deposit for the contents of processor register.

PUSH PSW (Push Processor Status word)

This push (saves) the combined 16 bit contents of the A-register and flags register on the top of the stack.

POP BC

- (i) Transfer the contents of the address given by SP to register C
- (ii) Transfer the contents of the address given by $SP+1$ to register B

Writing subroutine

* To first save the current contents of those registers which are used by the subroutine on the stack and then to restore the saved contents before the return instruction is given.

Parameter passing

— Parameters may be required to pass data both (i) to subroutine for processing and (ii) also for passing results back from the subroutine to the calling program after processing.

— Passing parameters to and from a subroutine is by means of a pointer to the start address in the memory where the parameters are stored.

pb 4

- (1) Initialise stack pointer at 2002
- (2) Store delay parameter in memory location 2020
- (3) Load delay time parameter into C register
delay parameter is 02

(4) call subroutine which is named TIMELY

(5) In subroutine, it consists of the following sequence,

(a) push (save) the combined 16 bit contents of the A register and flags register on the top of the stack
< push PSW >

(b) push (save) the contents of register C on stack
< push B >

(c) Read delay parameter from memory C

(d) Clear A

(e) compare the contents of C, if yes, goes to end otherwise proceeds

(f) Jump if the time is not set to zero

(g) Non operation stages \rightarrow 6 stages

(h) ~~execute~~ Execute delay instructions - decrement C register

(i) Jump loop

(6) ~~Transfer~~ Transfer the contents of the address given by

(a) SP to register B < Restore contents of register C >

(b) Transfer the contents of the given address given by SP to register A < Restore contents of register A >

(c) Return subroutine

Original program with subroutine

Additional for parameter passing

```

    MVI M, FF
    LXI SP, 2002
    LXI H, 2080
    MVI M, 02
    CALL TIMDLY
  
```

Initialise stack pointer
 Store delay parameters in memory location 2080
 call subroutine

```

    TIMDLY:
      PUSH PSW
      PUSH B
      MOV C, M
      MVI A, 00
      Loop:
        CMP C
  
```

Subroutine 1
 Save content of registers A, B, C on stack
 Read delay parameter from memory

```

    LXI H, 2081
    MVI M, FF
    CALL TIMDLY2
  
```

Store delay parameters in memory location 2081

```

    NOP
    NOP
    NOP
    NOP
    NOP
    NOP
  
```

Remove.

```

    DCRC
    JMP LOOP
    TIME:
      NOP B
      POP PSW
      RET
  
```

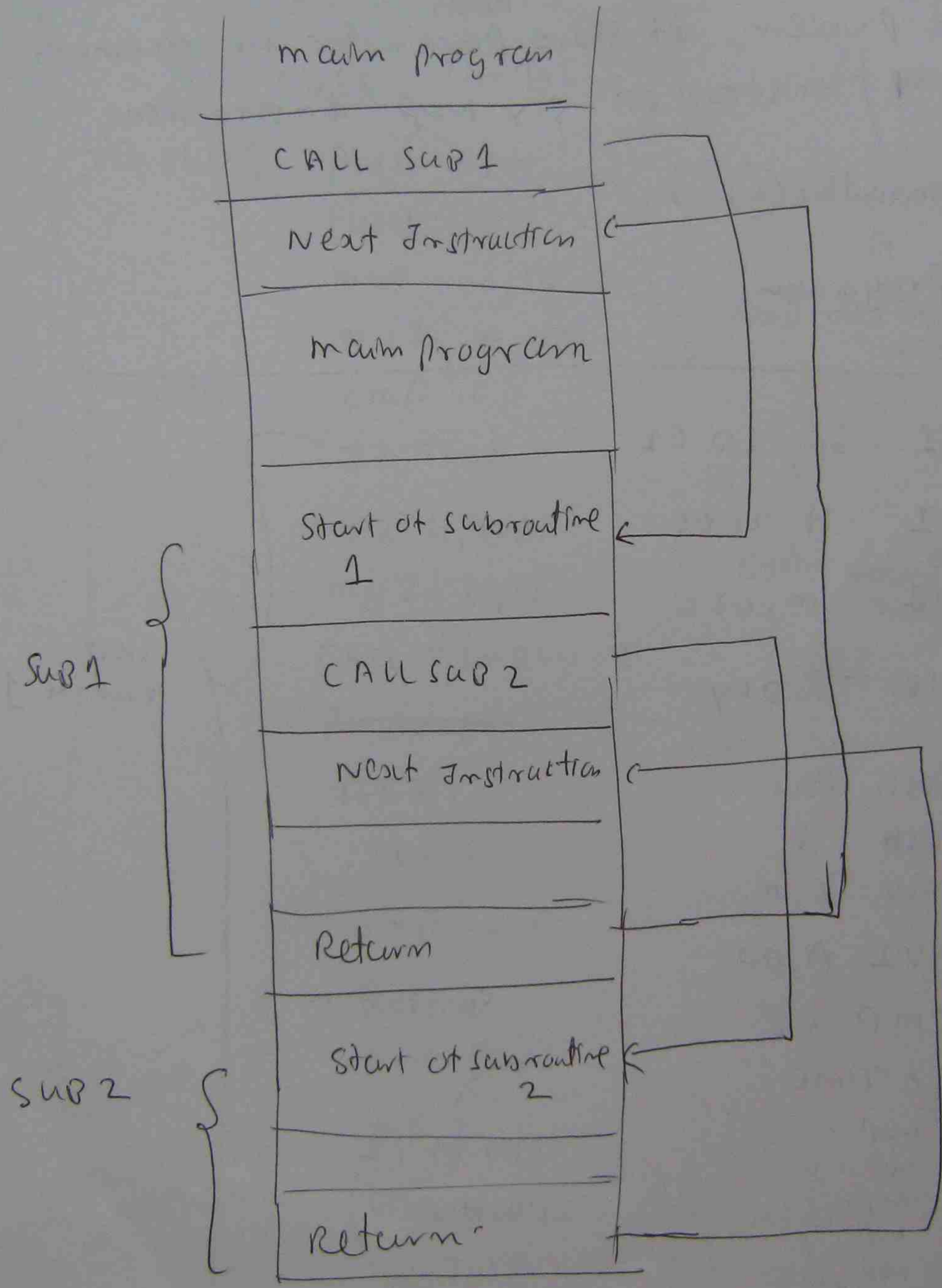
Restore content of registers A, B, C from stack
 Return subroutine

```

    Sub-routine 2:
      TIMDLY2:
        PUSH PSW
        RET
  
```

Nested subroutine

A sub routine calls another subroutine within itself



Pb 5

modification of pb 4

In above problem, it store ^{delay} parameter in memory location 20B0, instead of six nop instructions,

as subroutine (2)

write the program.

```
LXI SP, 20C2
```

```
LXI H, 20B0
```

```
MUI M, FF
```

```
CALL TIMDLY
```

MUI M, FF

Sub routine 1

TIMDLY

```
PUSH PSW
```

```
PUSH B
```

```
MOV C, M
```

```
MUI A, 00
```

Loop

```
CMR C
```

```
JZ TIME
```

Subroutine 2

```
LXI H, 20B1
```

```
MUI M, FF
```

```
CALL TIMDLY2
```

```
NOP
```

```
NOP
```

```
NOP
```

```
NOP
```

```
NOP
```

```
NOP
```

```
DCRC
```

```
JMP LOOP
```

```
POP B
```

```
POP PSW
```

```
RET
```

Time

TIMDLY2

```
PUSH PSW
```

```
RET
```

(GS)

main program

```

LXI SP, 2002
LXI H, 2000
MUI M, FF
CALL TIMDLY1

```

Initialise stack pointer

store delay parameter in memory location 2000

call subroutine 1

sub routine 1

```

TIMDLY1 PUSH PSW
        PUSH B
        MOV C, M
        MUI A, 00
Loop    CMP C
        JZ TIME

```

save contents of register A & C

Read delay parameter from memory

```

LXI H, 2001
MUI M, FF

```

store delay parameter in memory location 2001

```
CALL TIMDLY2
```

call subroutine 2

```

PUSH PSW
RET
    DEC C
    JMP LOOP
    POP B
    POP PSW
    RET

```

sub routine 2

```

TIMDLY2 PUSH PSW
        RET

```

As for TIMDLY1

except MUI instructions are replaced by

subroutine 2 call

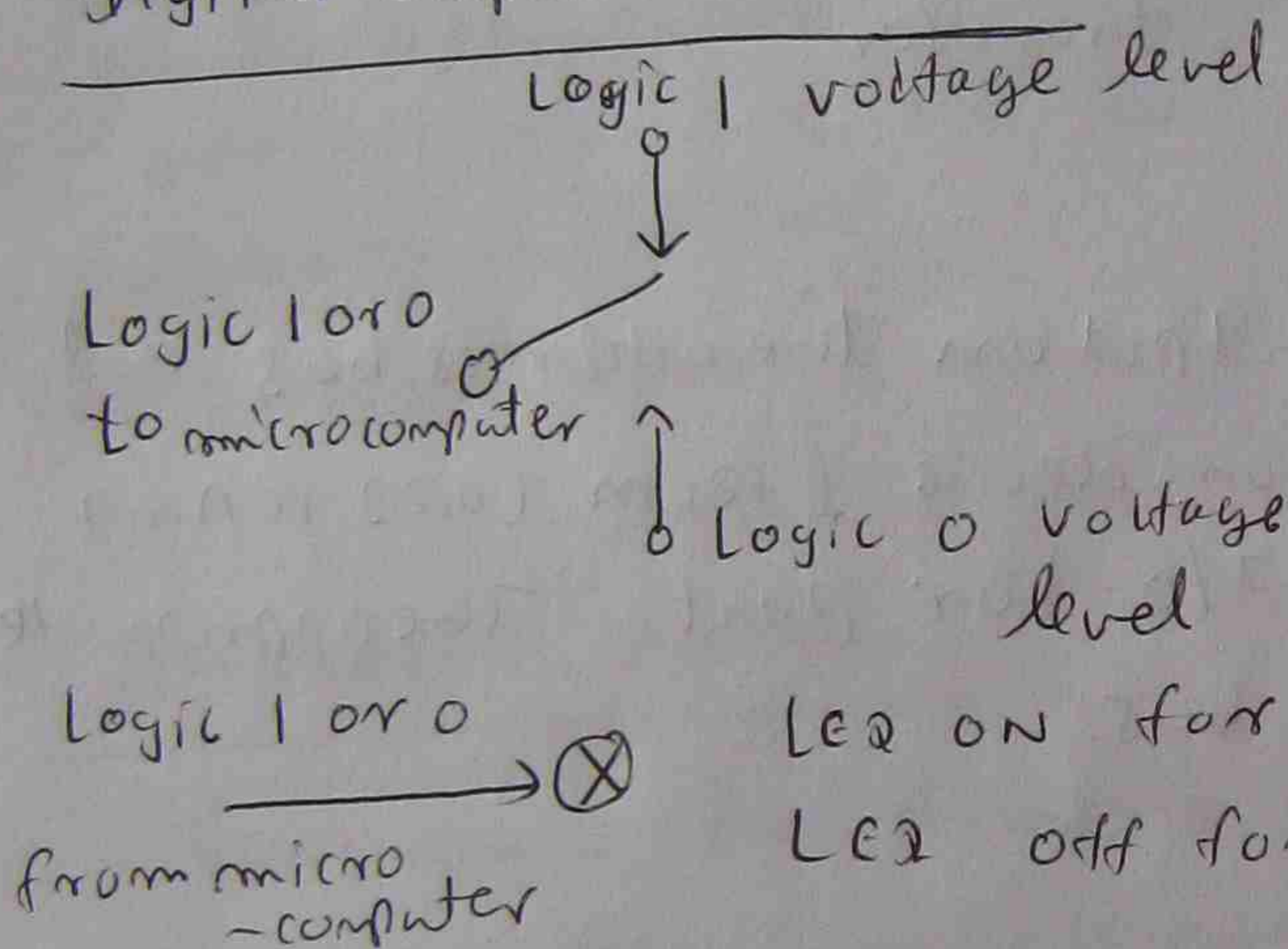
Exercise

- Look at questions in Exercise 9.1 to 9.7
- Study the answers provided for exercise 9.1 to 9.7
- Observe the way to find the solution
- Take practice with microprocessor drawing software

Digital Input and output

- A microcomputer is basically a digital component that can examine digital input signals and perform functions as a consequence of these inputs to yield digital output signals.
- The external devices outside the micro computer produce (or) accept signals which are not necessarily ~~in~~ digital in nature, special interface circuitry is often required to transform these external signals into a form suitable for the microcomputer

Digital Input and output

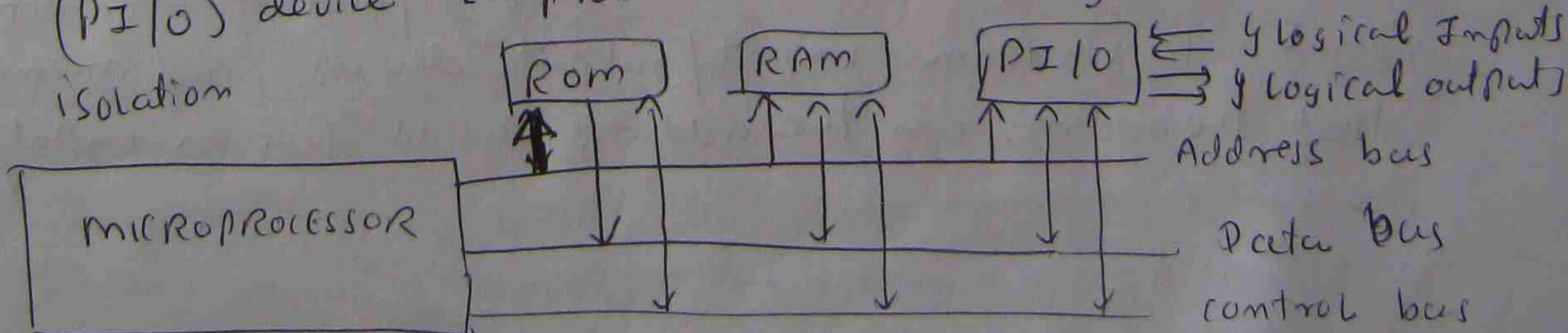


A simple digital input to a microcomputer can be produced by a single pole switch.

Logic 0, 1 depends on switch position

LED ON for logical 1
LED OFF for logical 0

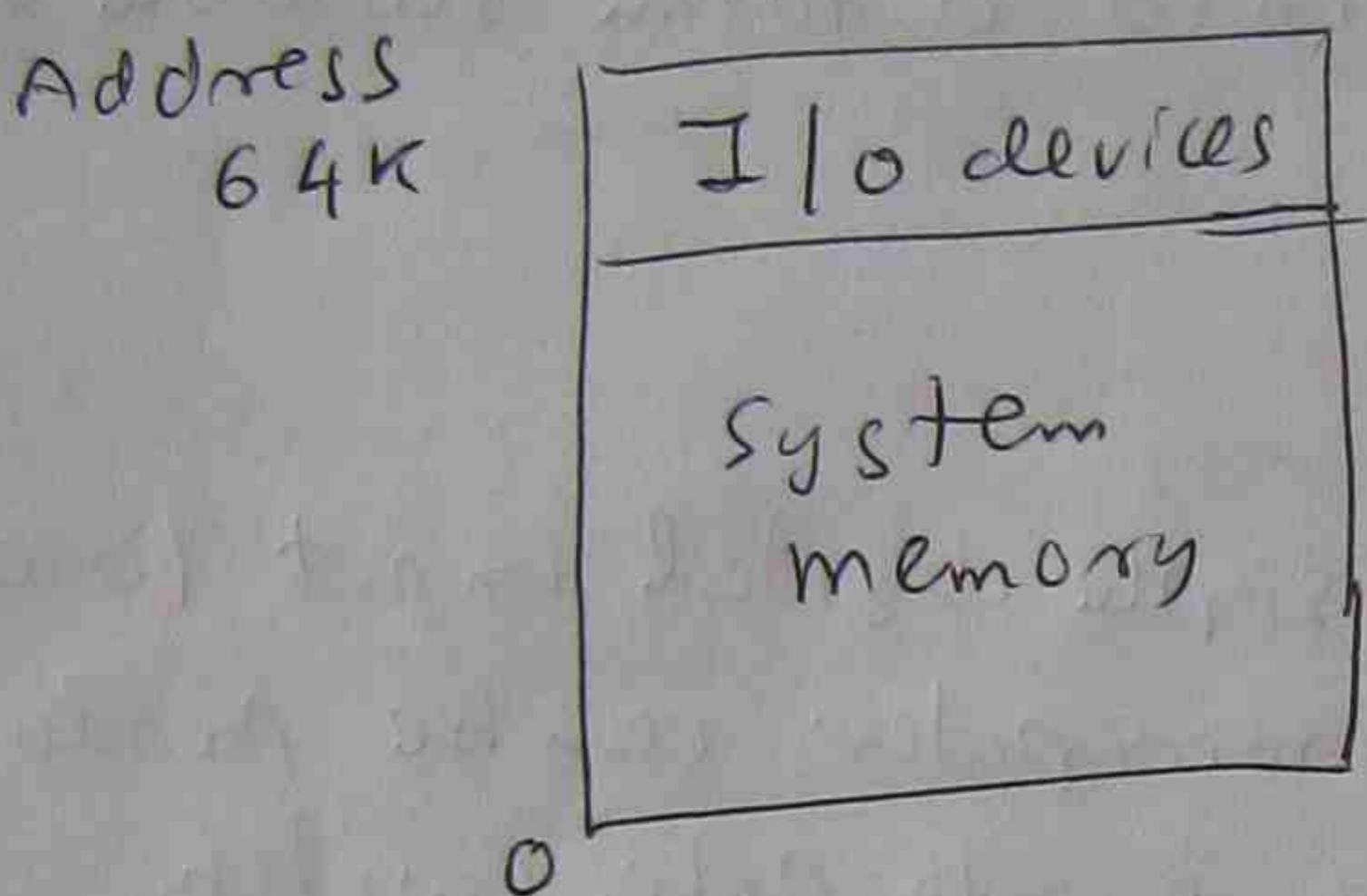
- The information intended for an output indicator must be latched by a suitable circuit.
- The processor can then send data to the output latching device which captures the data at the appropriate time determined by bus control signals and then provides a continuous output until ^{new} data is sent to it.
- The system incorporates a programmable input/output (PI/O) device to provide the necessary latching and isolation



data transfer of input/output data between a microcomputer bus and input/output device

- memory mapped input/output
- programmed input/output.

i memory mapped input/output

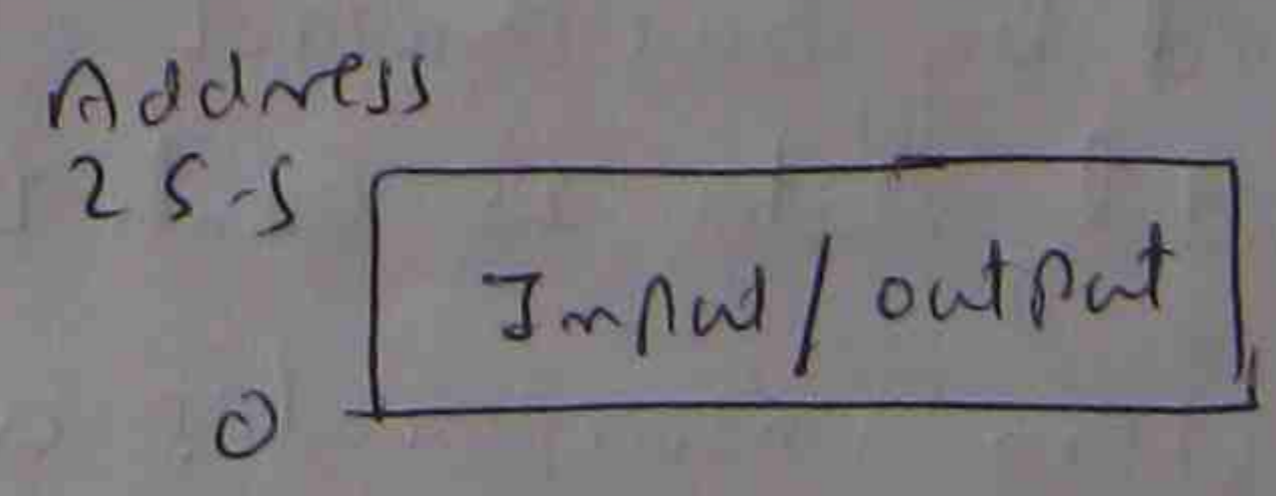
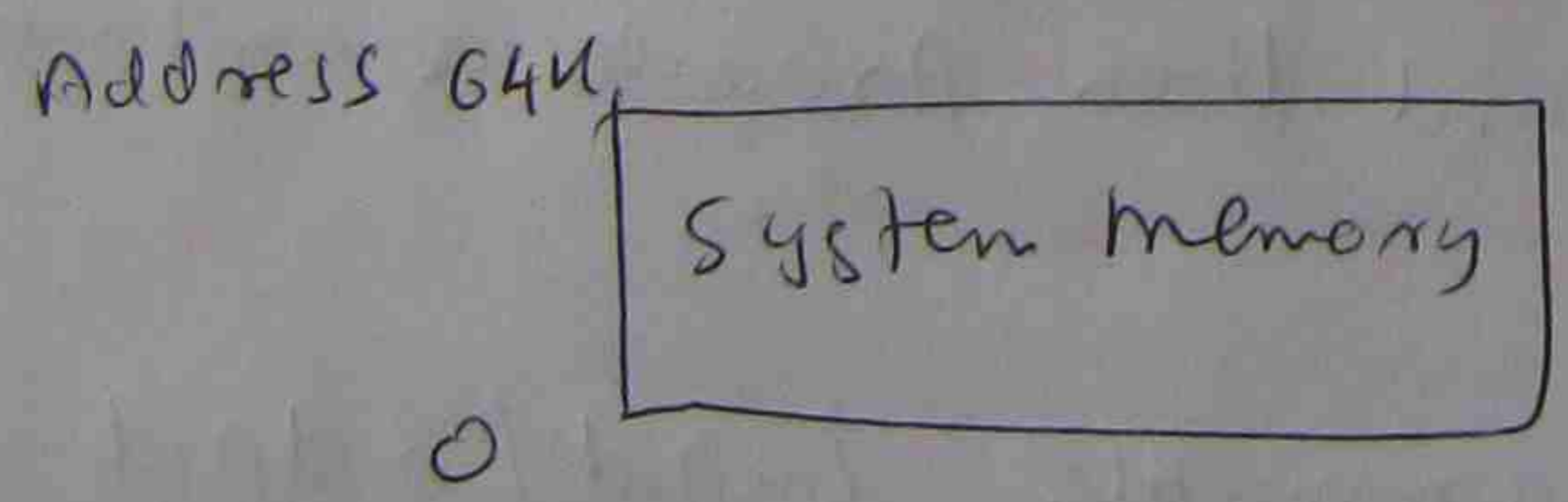


- The same instructions are used for both memory and input/output data transfer

The appropriate address is output on the address bus and recognized either by a memory device (ROM (or) RAM) or input/output device (PIO) or port. The appropriate data is transferred on data bus.

ii Programmed input/output

- Input/output data transfers are accomplished by means of special instructions executed by the processor - IN and OUT for the Intel 8085.



- microprocessor generates an input/output request signal to inform input/output devices (and memory) that the address on the address bus is for an input/output device.

iii Programmable input/output

Digital input and output in most microprocessors is controlled by programmable input/output devices and programmed input/output is normally used. A PIO device can control a number of individual input and output lines. These are normally grouped in to a number of ports, each comprised of eight lines which may be programmed to operate either as inputs or as outputs.

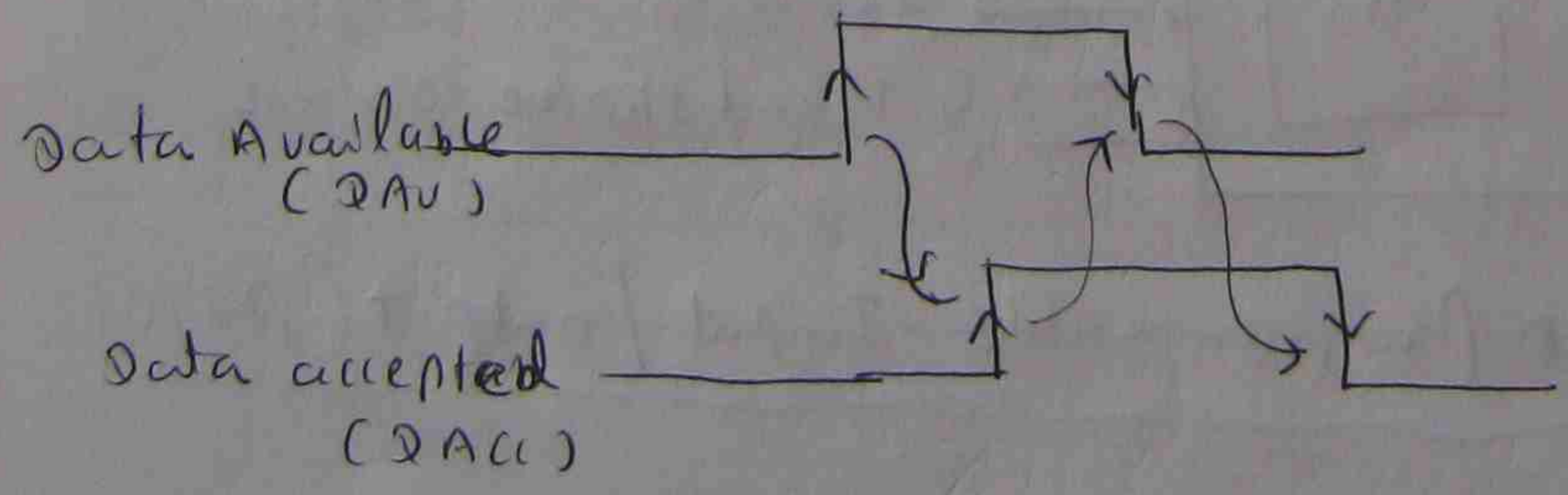
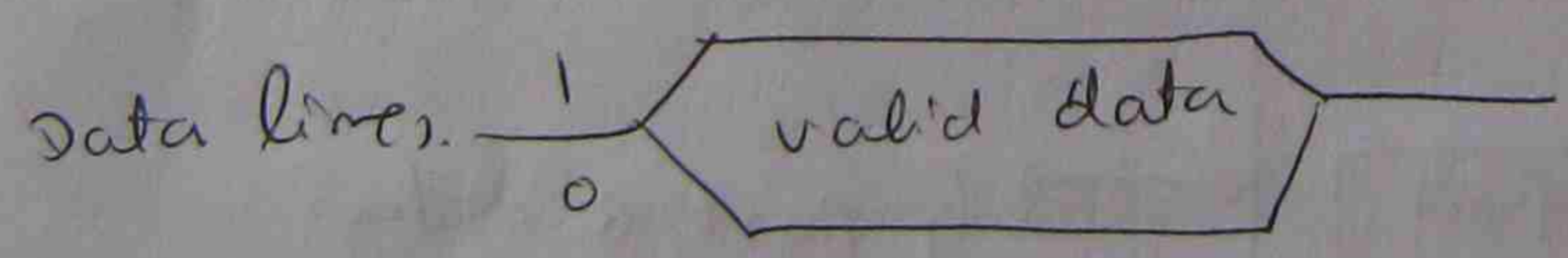
Steps

- write appropriate command information into a specific addressable registers within the device when the system is being initialised
- data is read ^{from} (or) written to a port.

Handshake control

Synchronise the transfer of data between PIO and external device and consequently most PIOs provide control lines for this function.

Handshake - Typical transfer sequence.



- Sending line places data on data line
- Data available (DAV) line is set
- Receiving device detects the setting on DAV line

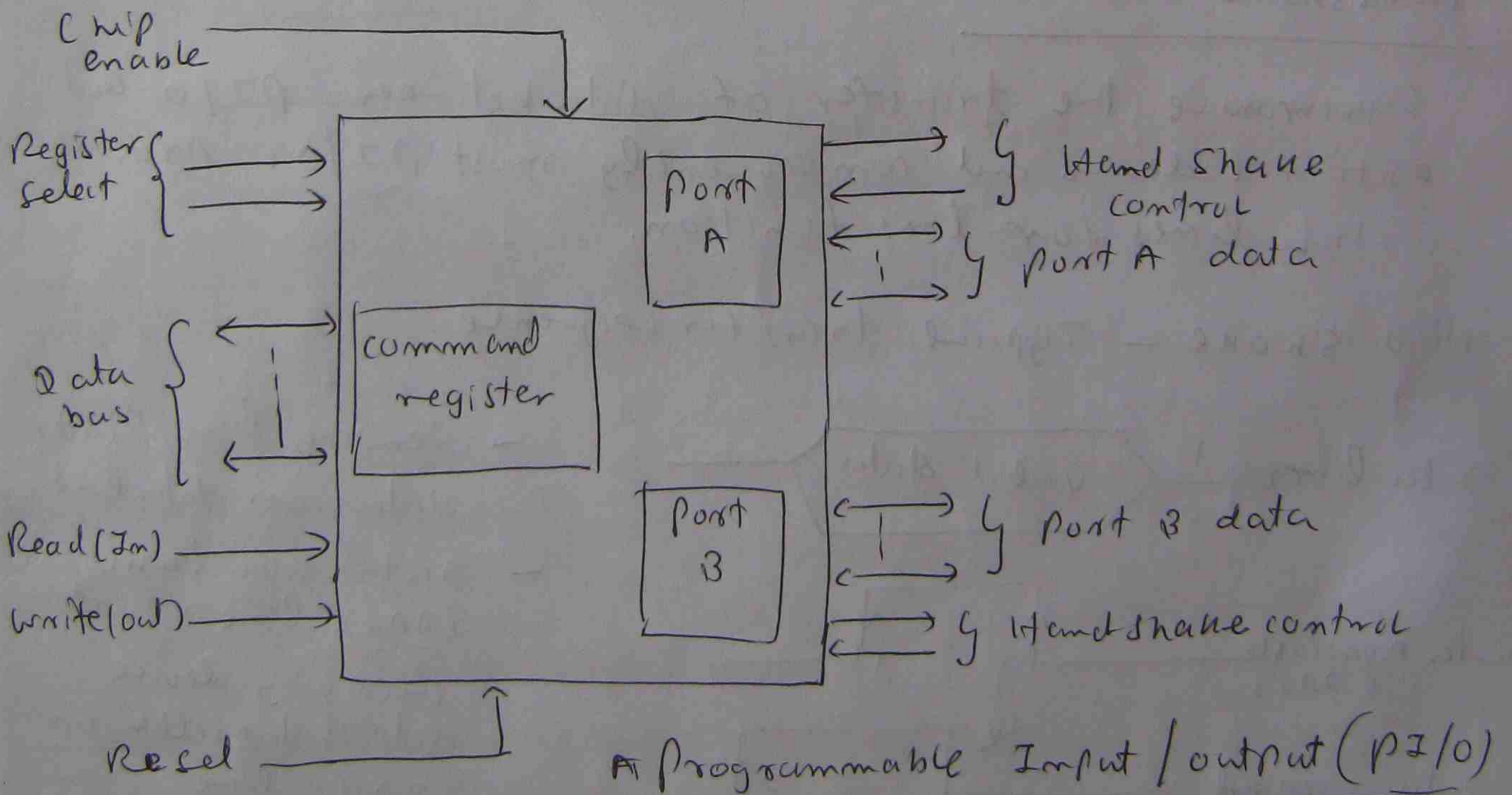
- Then responds by setting the data accepted "DACC" line
- Sending device interprets the setting on DACC line

(70)

- Receiver detects that \overline{DAV} line has been reset
- Reset \overline{DAV} line to permit further data transfer.
- There is a chip enable input on all the devices which are connected to the microprocessor bus.
- RAMs, ROMs, PI/Os - They are used to ensure that only one device responds to each data transfer on the bus.
- Port A, B select the appropriate register within PI/O itself - command.

Port Initialisation

Some microprocessor, each programmable input/output device is a separate integrated circuit but in others, it is incorporated into other system components.



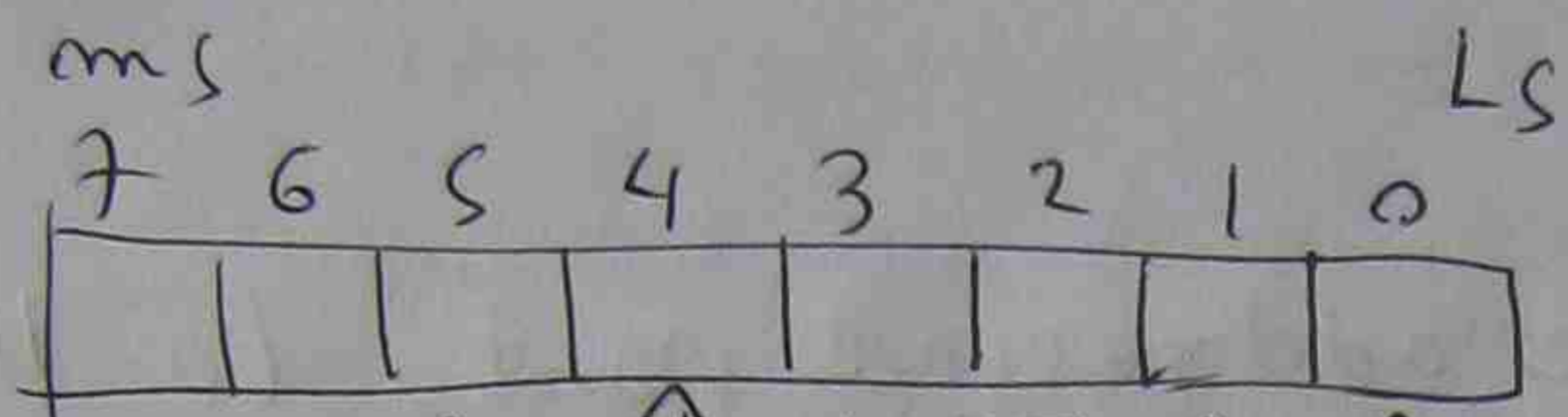
Command Information

- Initialise the port, timer and input/output ports are programmed together

OUT address

(Transfer the contents of the processor A register to the addressed input/output device)

8155 command byte



Port A { 0 = Input
1 = output

Port B { 00 = Input
01 = output
10 = hand shake
11 = Interrupt control

Port A interrupt { 1 = enable
0 = disable

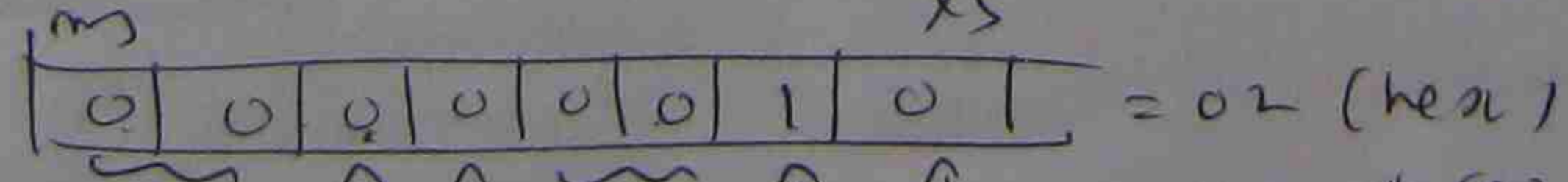
Port B interrupt

Timer command
00 = do not affect timer
11 = set for continuous operation.

```
MUI A, 02
OUT 20
```

- Transfer the command data 02 (hex) to the command register in selected 8155
- configure port A to be 2 inputs
- B 8 outputs
- C second input port

Timer is not affected



Do not affect timer
Disable interrupt on port B

Port (A) Input
Port (B) output
Port C Input

Disable Interrupt on Port A

Address (hex)	Port / Register
20	Command / Status register
21	Port A
22	Port B
23	Port C

(72)
 Typical
 Port /
 register
 addressing

IN address

(Transfer data from the addressed port to the A - register)

Ex ① Write the Program

- Port A is first configured as input port
 ————— B ————— output —————
 at port 20

Data is input from port A & output to port B at port 22

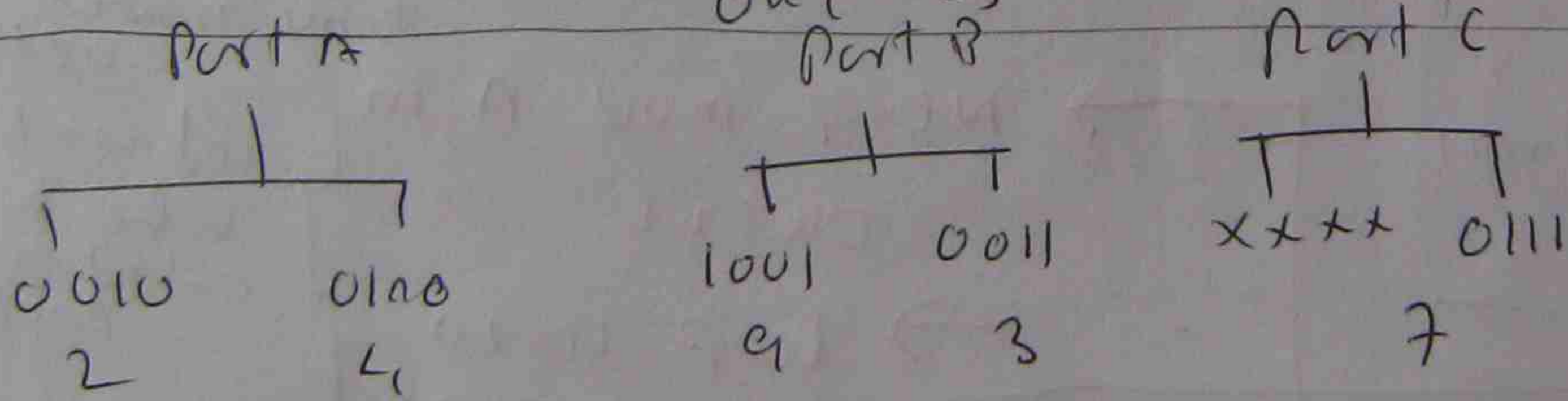
Process is repeated continuously

Assembly Instruction	Comments
MVI A, 02	Load command byte in A
OUT 20	Load command register
START IN 21	Read data from Port A
OUT 22	Output data to port B
JMP START	Repeat

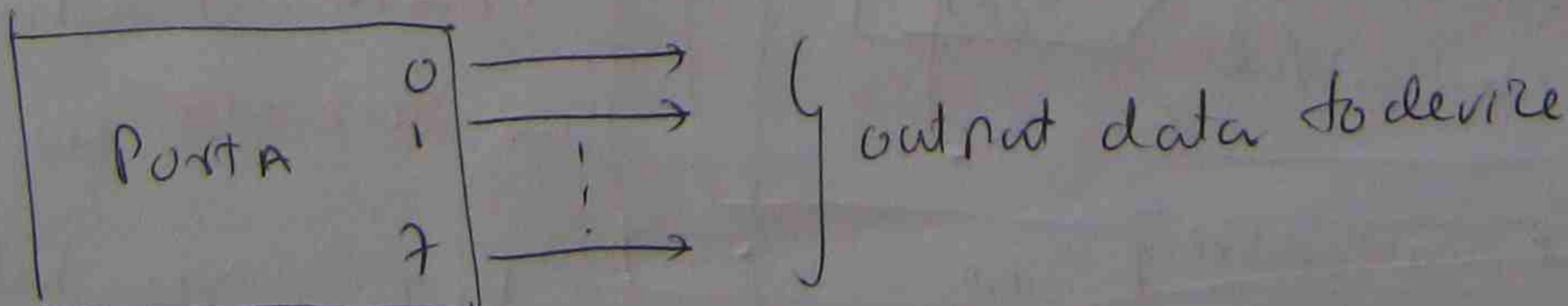
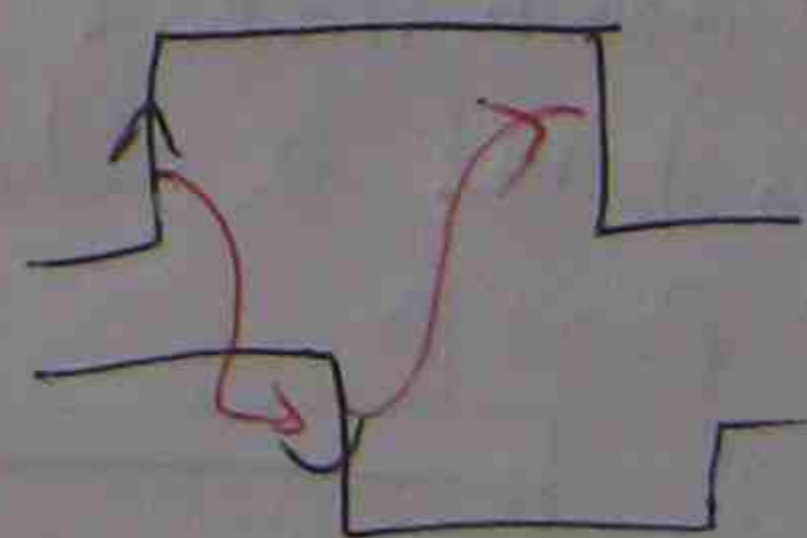
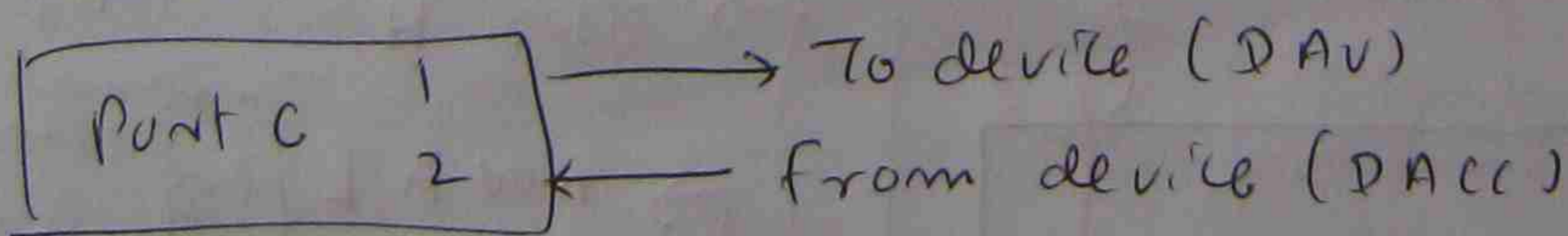
Ex 2 Initialize port A, B, C as output ports.
Then output decimal number 24937 in BCD form.

- Step
- Load command byte in A
 - Load command register
 - output 24 to port A at line 21
 - output 93 to port B at line 22
 - output 07 to port C at line 23

Assembly instruction	Comments
<code>mvi A, 07</code>	Load command byte in A
<code>out 20</code>	Load command register
<code>mvi A, 24</code>	output 24 to port A
<code>out 21</code>	
<code>mvi A, 93</code>	output 93 to port B
<code>out 22</code>	
<code>mvi A, 07</code>	output 07 to port C
<code>out 23</code>	



Handshake control



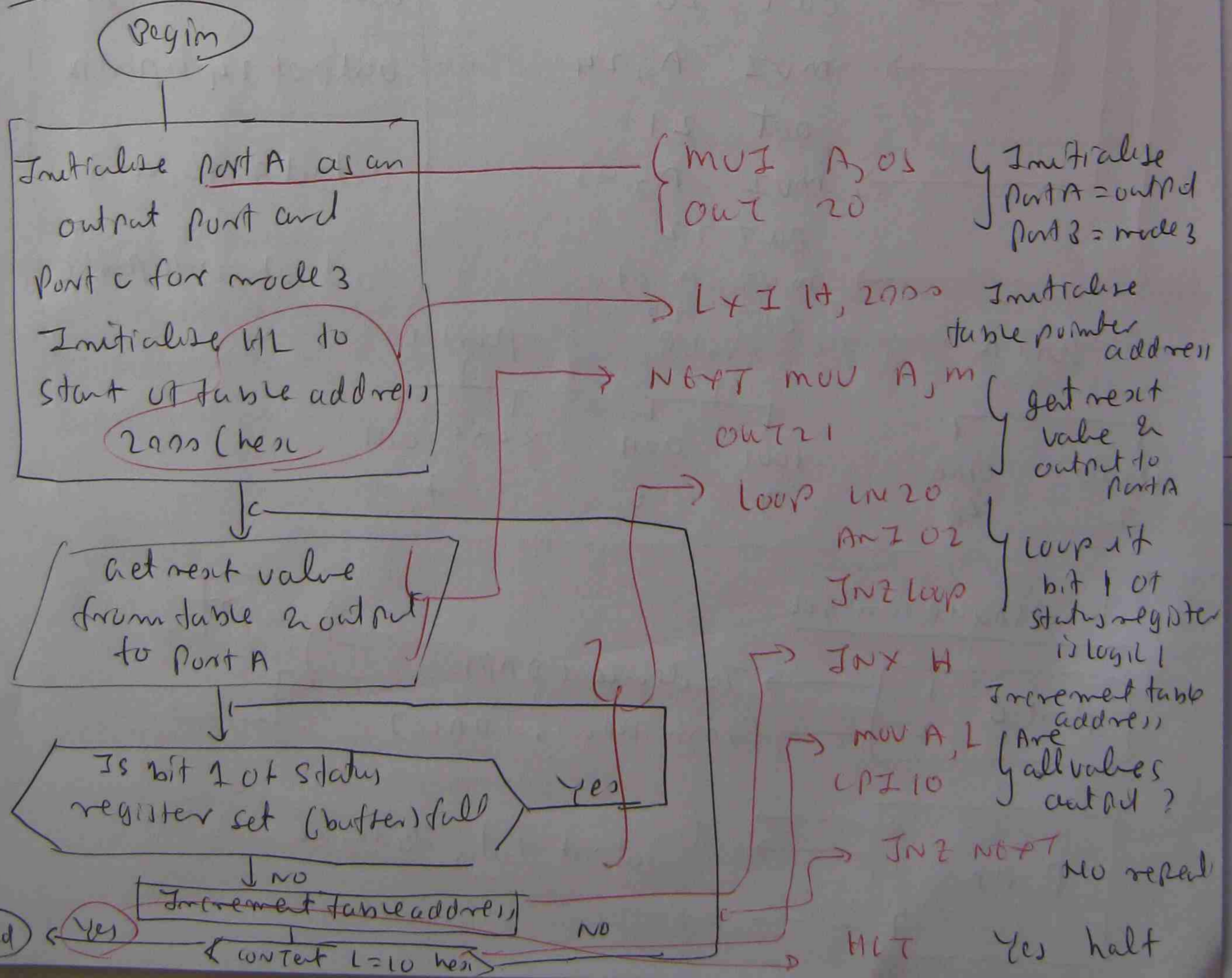
~~3/6/9/11~~

Ex 3

Handshake control lines of port c to output 16 values from a table in memory - starting address 2900 hex to an external device connected to port A.

- port c has been initialised to operate in mode 3 (hand shake mode)
- After data has been output to port A bit 1 of port c will automatically go to logic 1, indicating to the external device that new data available.

Flow

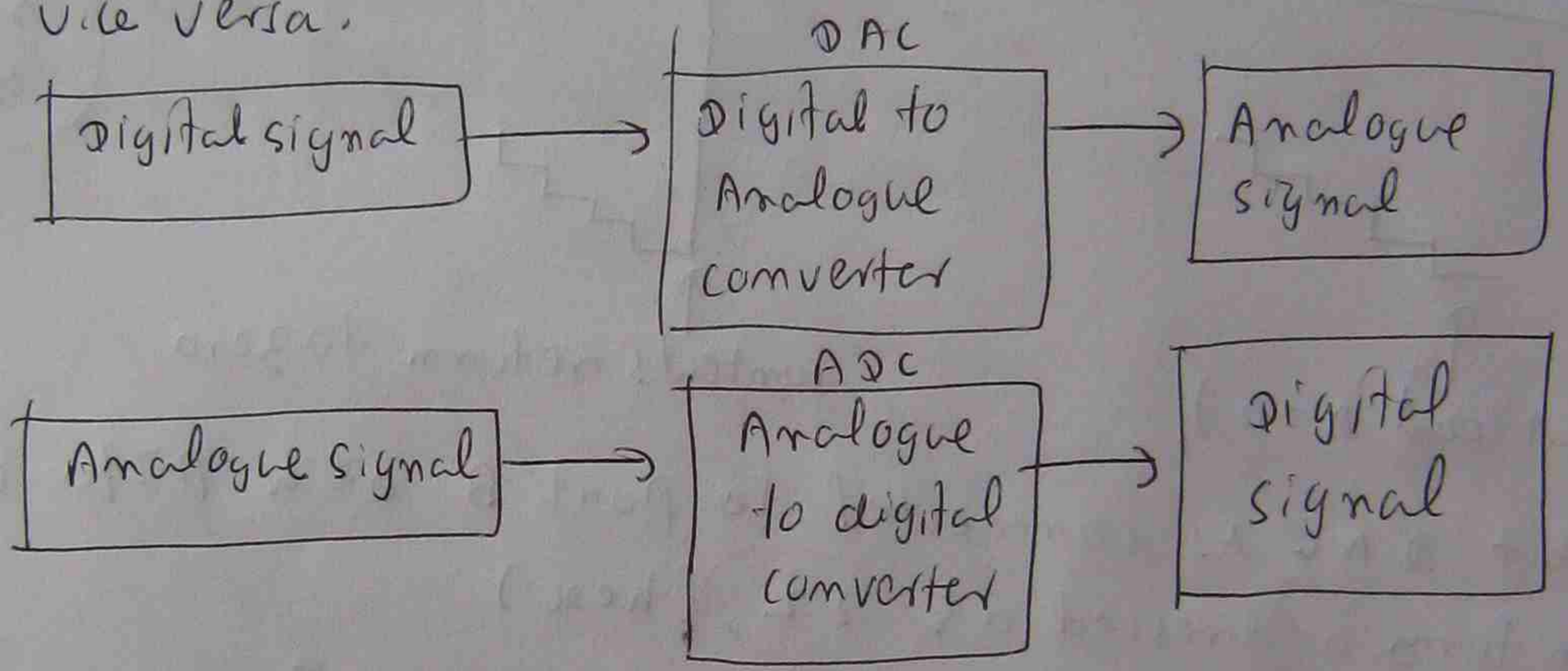


(75)
WGEN (7) ANALOG INPUT/OUTPUT & INTERRUPT

Input data - varying analogue signal, output voltage from a temperature transducer.

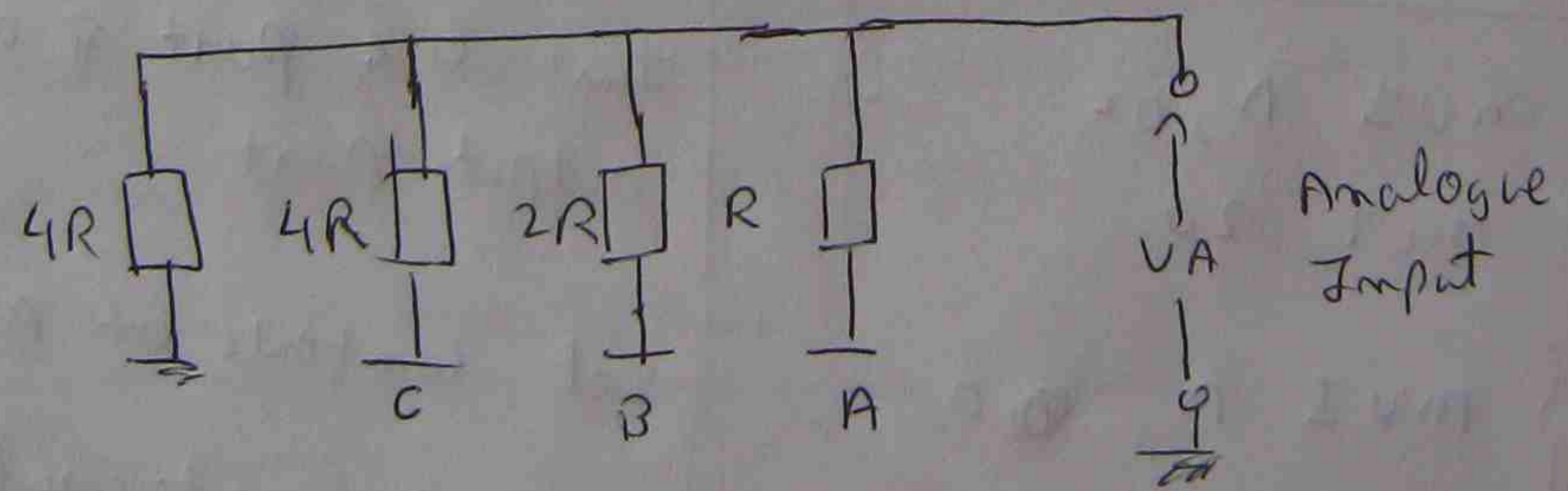
Output data - The output data from the microprocessor is often required in analogue form, for example to drive a motor.

It is necessary to have additional interface circuitry between the input/output ports of the microprocessor and the controlled peripheral devices, both to convert analogue signals in to digital form and vice versa.



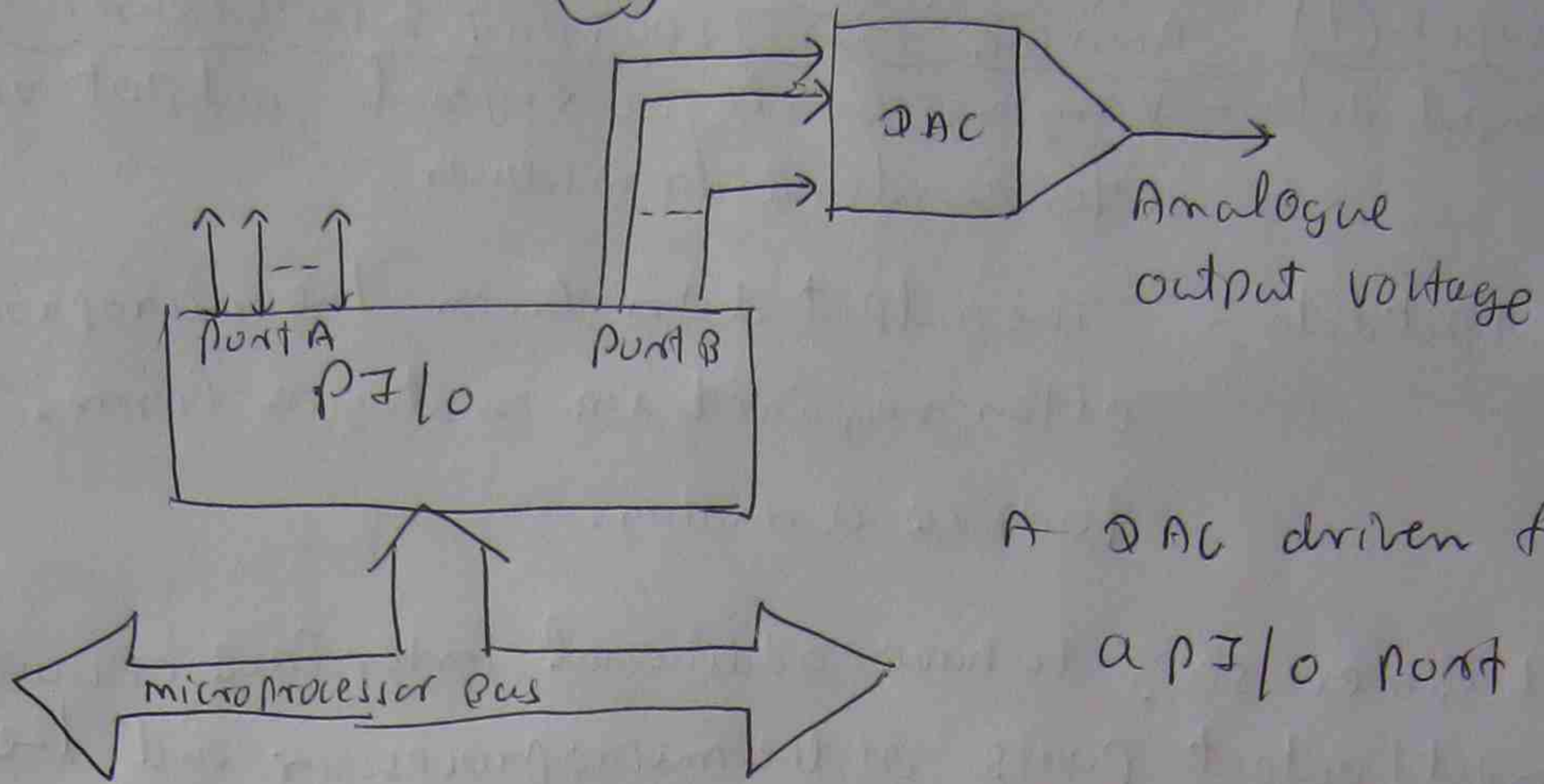
Digital to analogue conversion

A digital number can be converted to an analogue voltage by selectively adding voltages which are proportional to the weighting of each binary digit.



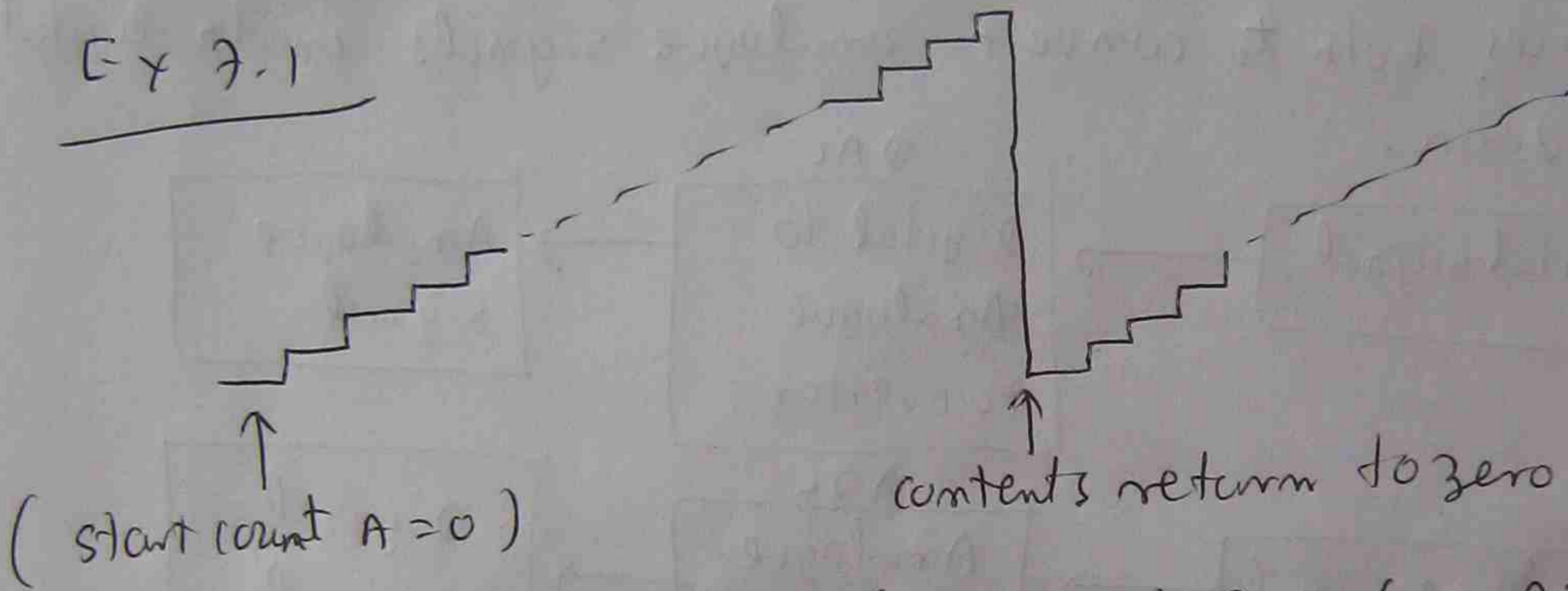
A	B	C	V_A
0	0	0	0
0	0	1	$V/8$
0	1	0	$V/4$
0	1	1	$3V/8$
1	0	0	$V/2$
1	0	1	$5V/8$
1	1	0	$3V/4$
1	1	1	$7V/8$

76



A DAC driven from a P/I/O port

Ex 7.1



Sawtooth generation

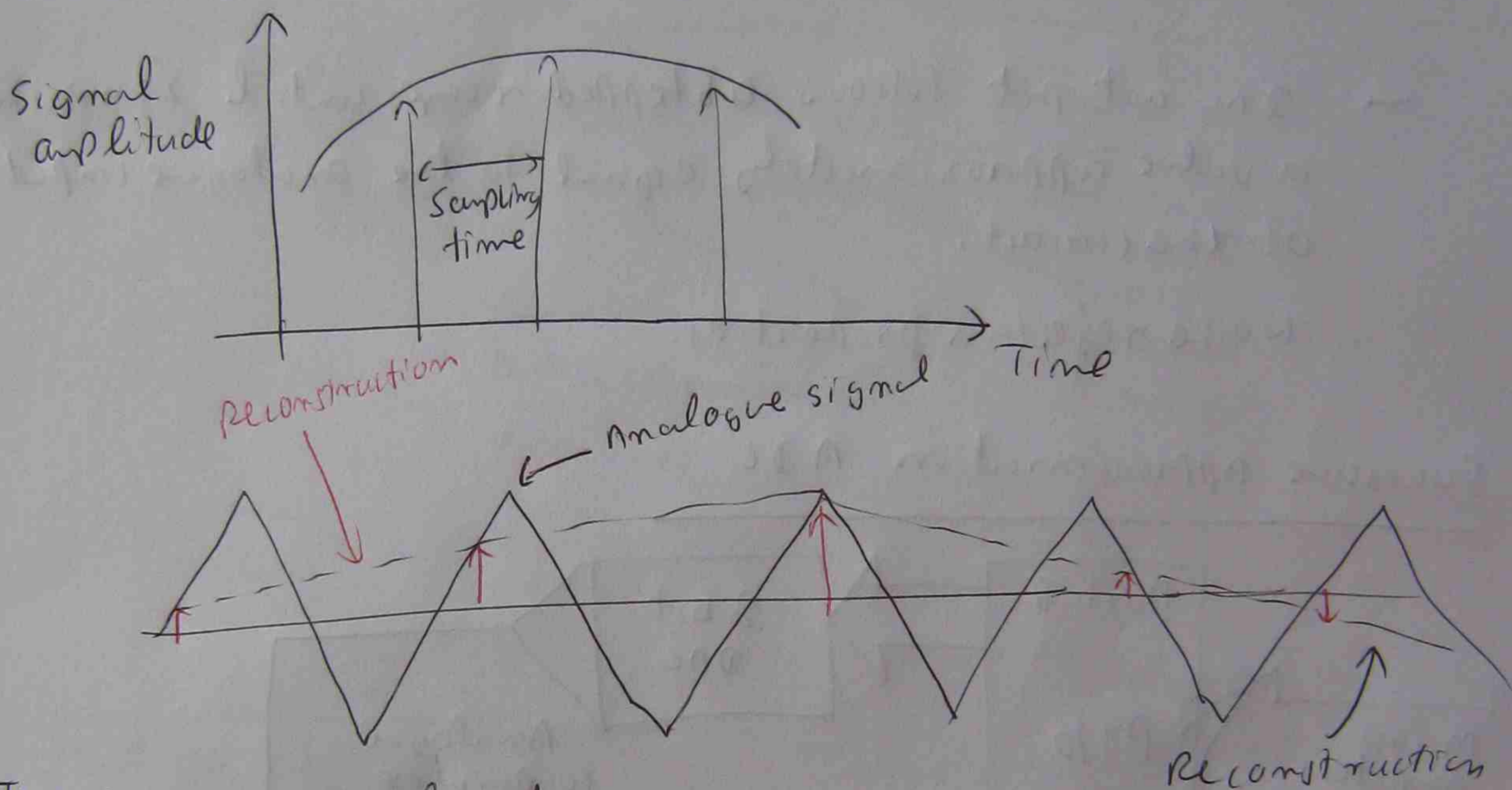
8 bit DAC is connected to port B of a P/I/O which is in turn addressed as 22 (hex)

A sawtooth wave form is then readily generated by using the A-register as a counter. & outputting its contents after each increment. Write a program

Assembly Instructions	Comments
<pre> MVI A, 02 OUT 20 </pre>	Initialize port B as an output port
<pre> MVI A, 00 </pre>	Set contents of A to zero
<pre> COUNT OUT 22 </pre>	Output current count
<pre> INR A </pre>	Increment count
<pre> JMP COUNT </pre>	Loop back

Analogue to digital conversion

The conversion of an analogue signal to a digital number implies a process of signal sampling. A digital number can only accurately represent a changing analogue signal for a short period of time.

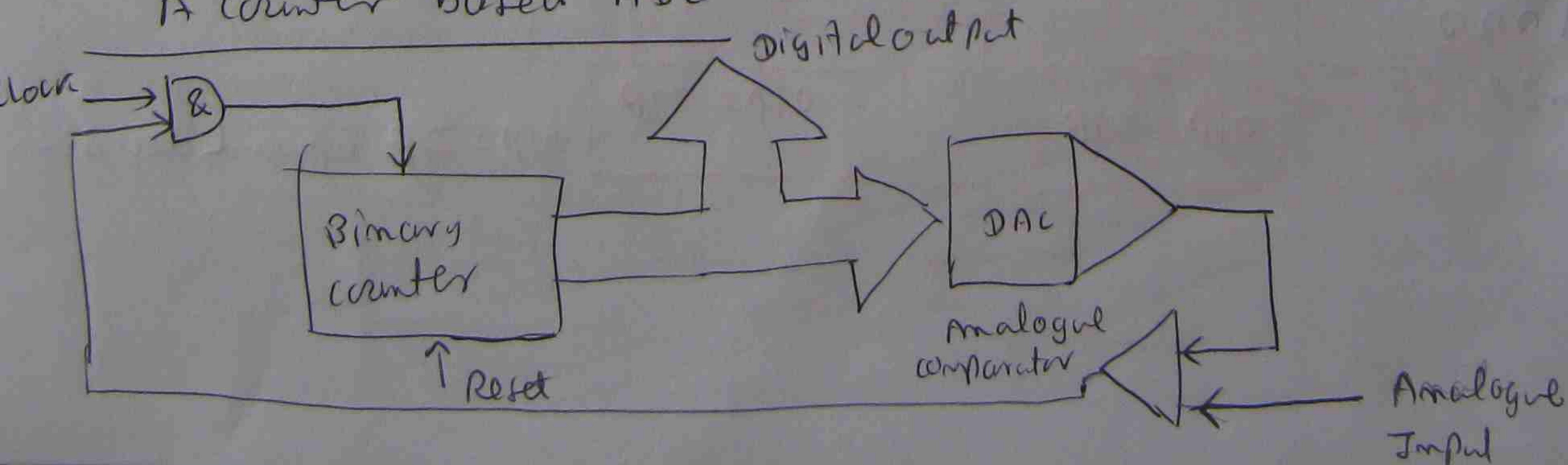


The Shannon sampling theorem

It is possible to severely distort the digital representation of an analogue signal by sampling it too infrequently.

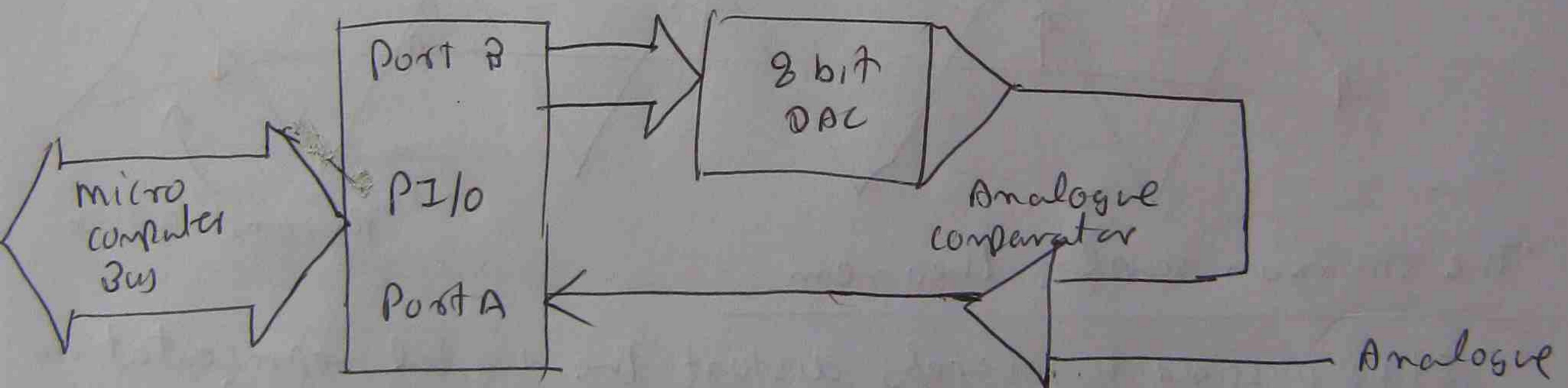
The Shannon sampling theorem provides that an analogue signal can be completely reconstructed if it is sampled at a uniform rate greater than twice the higher frequency component of the original signal.

A counter based ADC



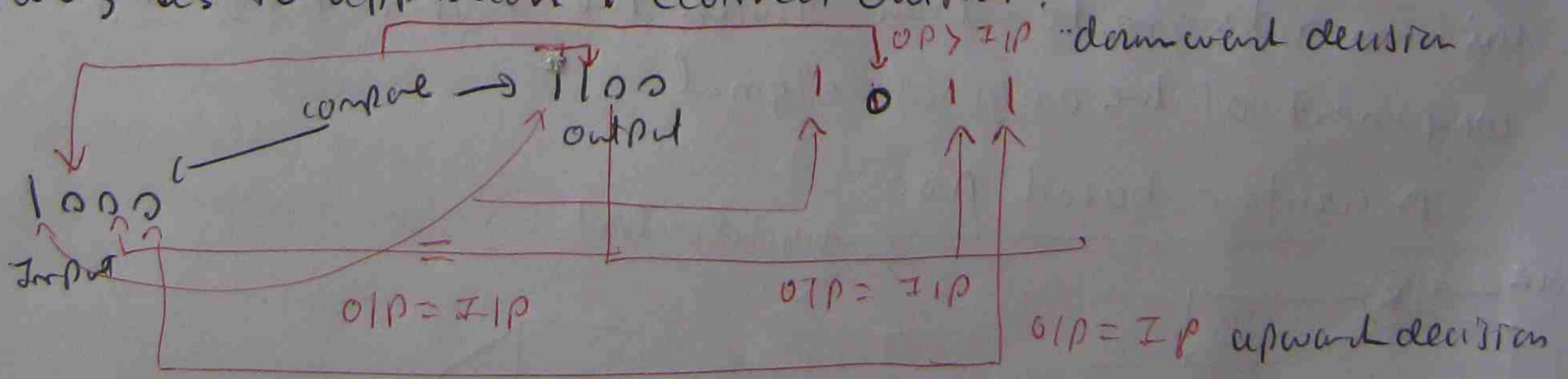
- The counter is initially reset at the start of a conversion.
- until DAC output just exceeds the analogue input, counter clock pulses are then enabled, causing the comparator to further counter clock pulse.
- Digital representation of the analogue input is then binary output of the counter
- DAC output follows a stepped ramp until it reaches a value approximately equal to the analogue input of the circuit.
- noise rejection properties.

Successive Approximation ADC

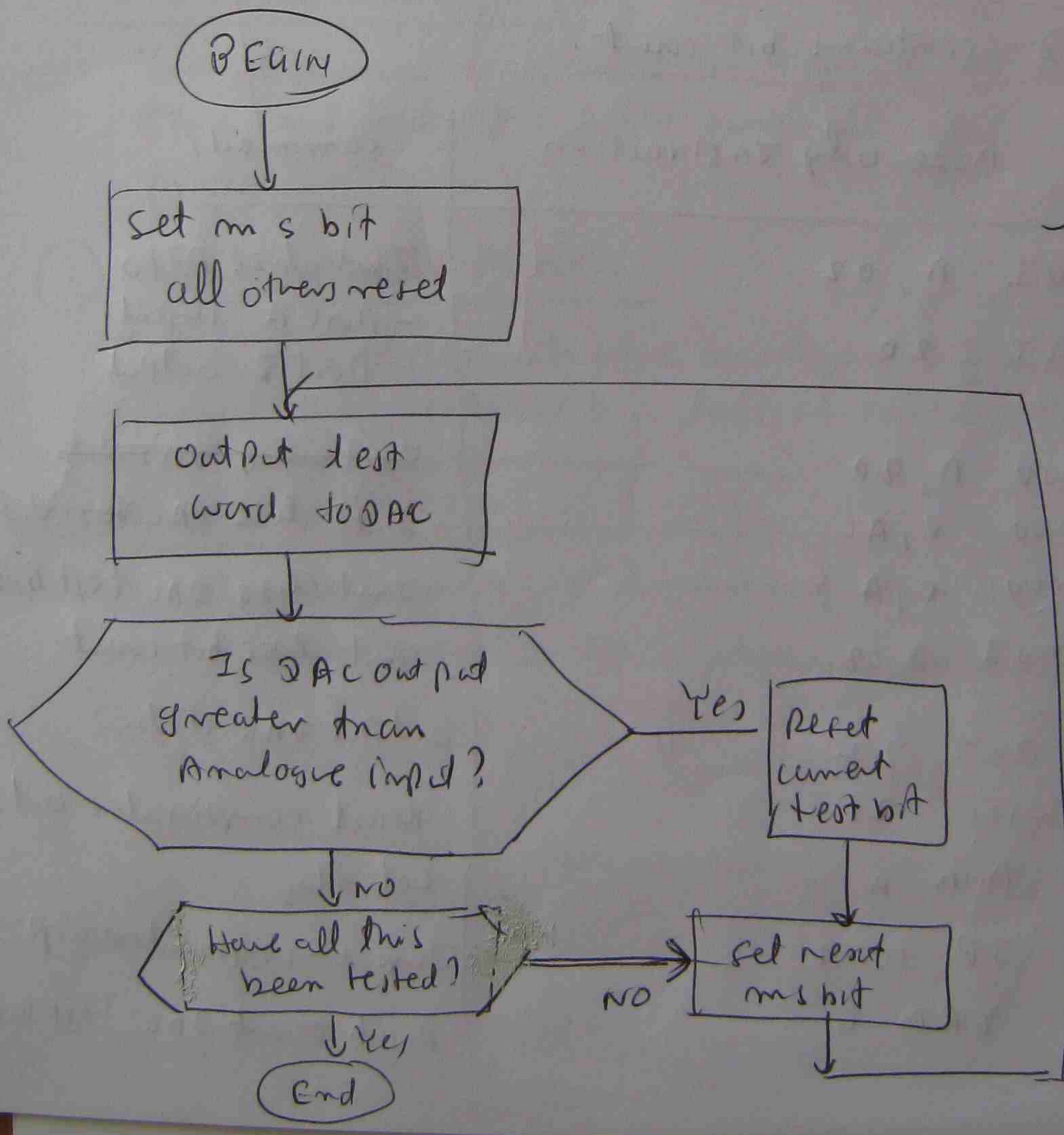
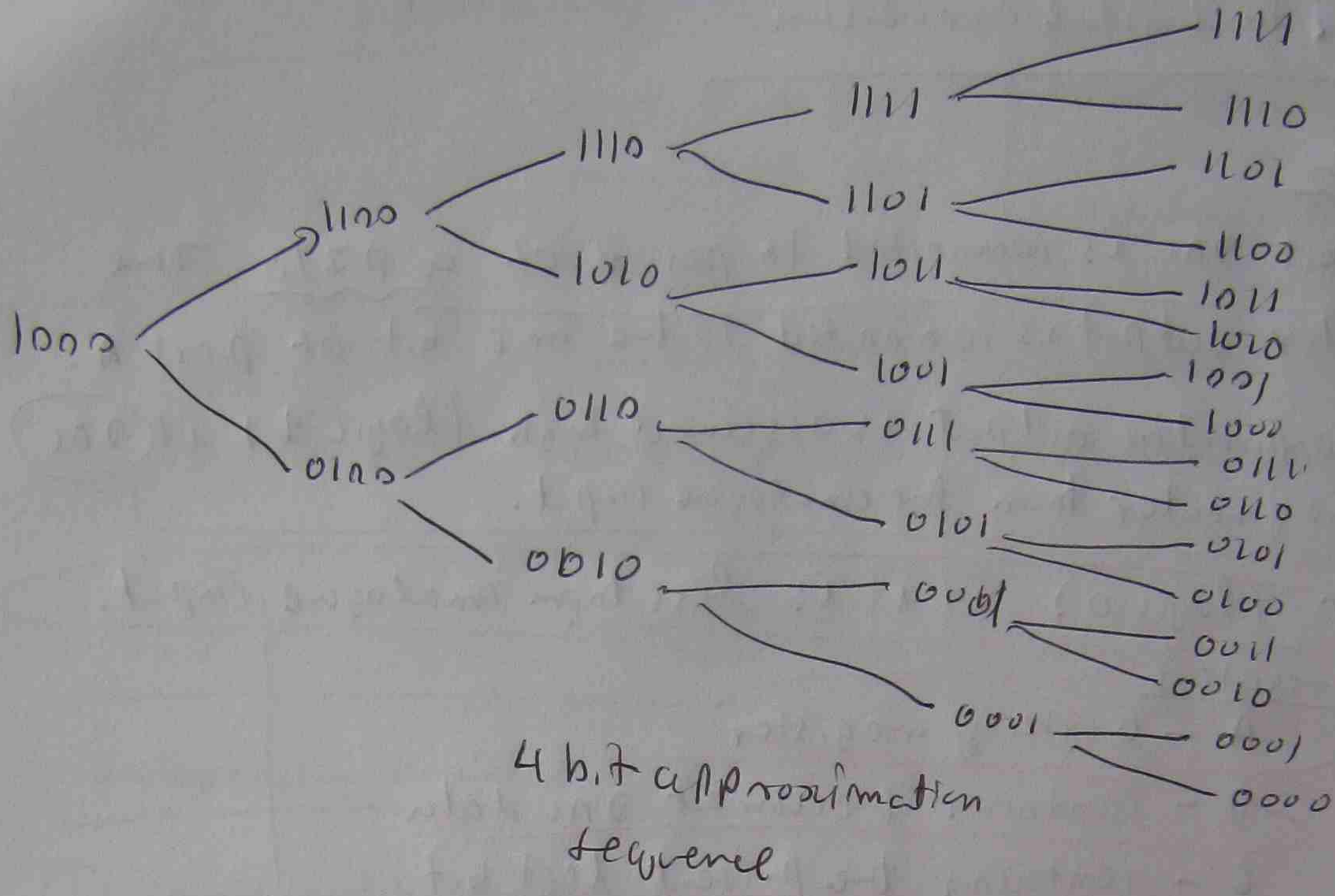


Hardware for a software based ADC

The digital output altering this value each time in such a way as to approach the correct output.



79



- set most significant bit of DAC input to 1
- Then accessible bits in order of significance
- output is examined
- If it is greater or less
- If less, the bit being tested is held at 1 & the next bit is test

Analogue to digital conversion

Ex 7-2

8 bit DAC is connected to port B of a P/I/O. The comparator output is connected to the msb bit of port A.

The comparator output is assumed high (logic 1) if DAC output is greater than the analogue input.

low (logic 0) if it is less than analogue input.

Processor registers

A - working register

B - contains the current DAC data

C - contains the present test bit

D - contains bit count.

Assembly Instruction

Comments

MVI A, 02

OUT 20

MOV A, B0

MOV B, A

MOV C, A

MVI D, 08

Initialize P/I/O

- Port A input

- Port B output

~~Initialize DAC data~~

Initialize DAC test data

Initialize DAC test bit

Initialize bit count

RGPGAT

OUT 22

IN 21

ANA A

JP COM7

XRA C

output DAC data

Read comparator output

Set flag

Jmp if comparator output = 0

Reset count DAC test bit

Assembly Instruction	Comments
comz mov B, A	save DAC data
mov A, C	} update DAC test bit
RAR	
mov C, A	
ORA B	
DEC D	decrement bit count
JNZ REPEAT	Jump if not zero

Interfacing analogue devices

① Sample and hold circuit

These are used to sample a signal at a precise time and hold the value constant during the conversion process

② Analogue multiplexers

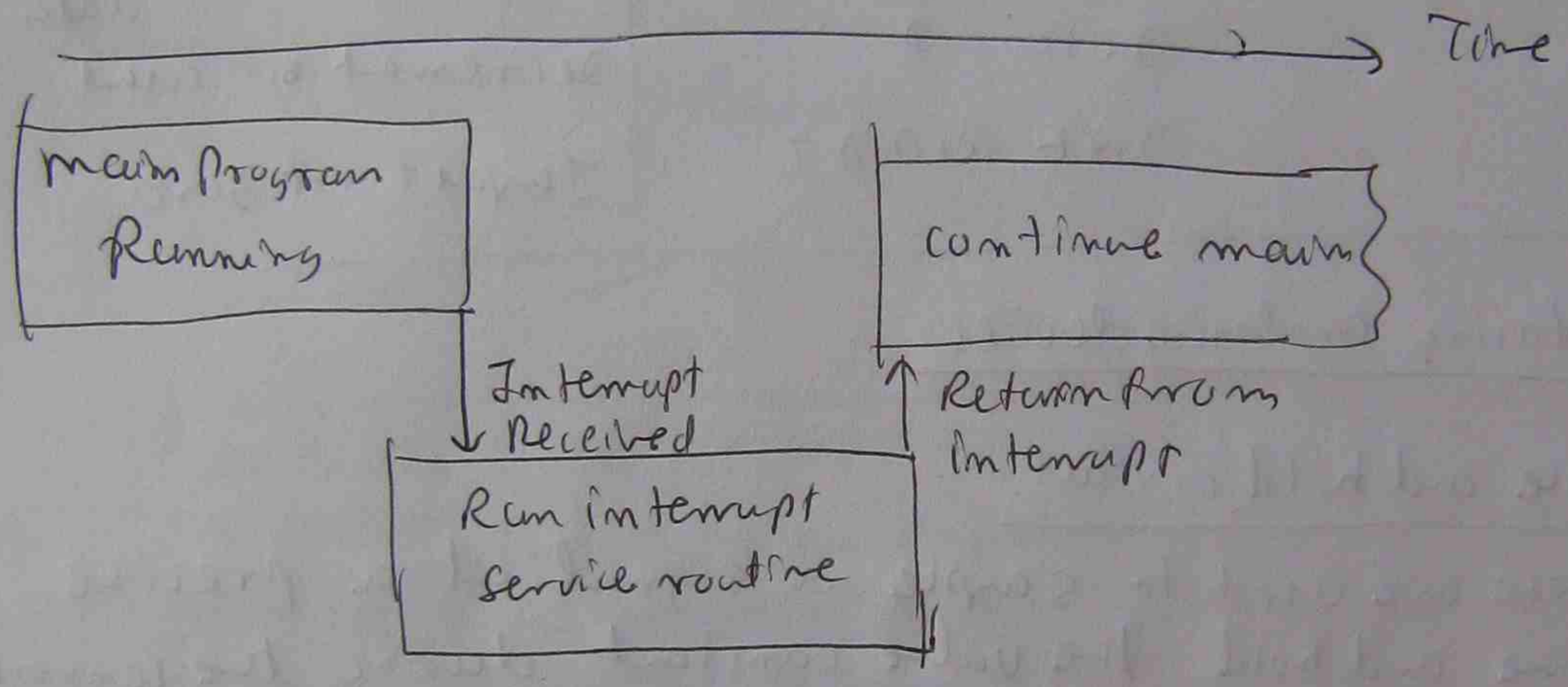
These devices permit one analogue signal out of several to be selected by logical control signals.

③ Real time clock

Signal sampling and construction is often performed in conjunction with an interrupt driven real time clock.

Interrupts

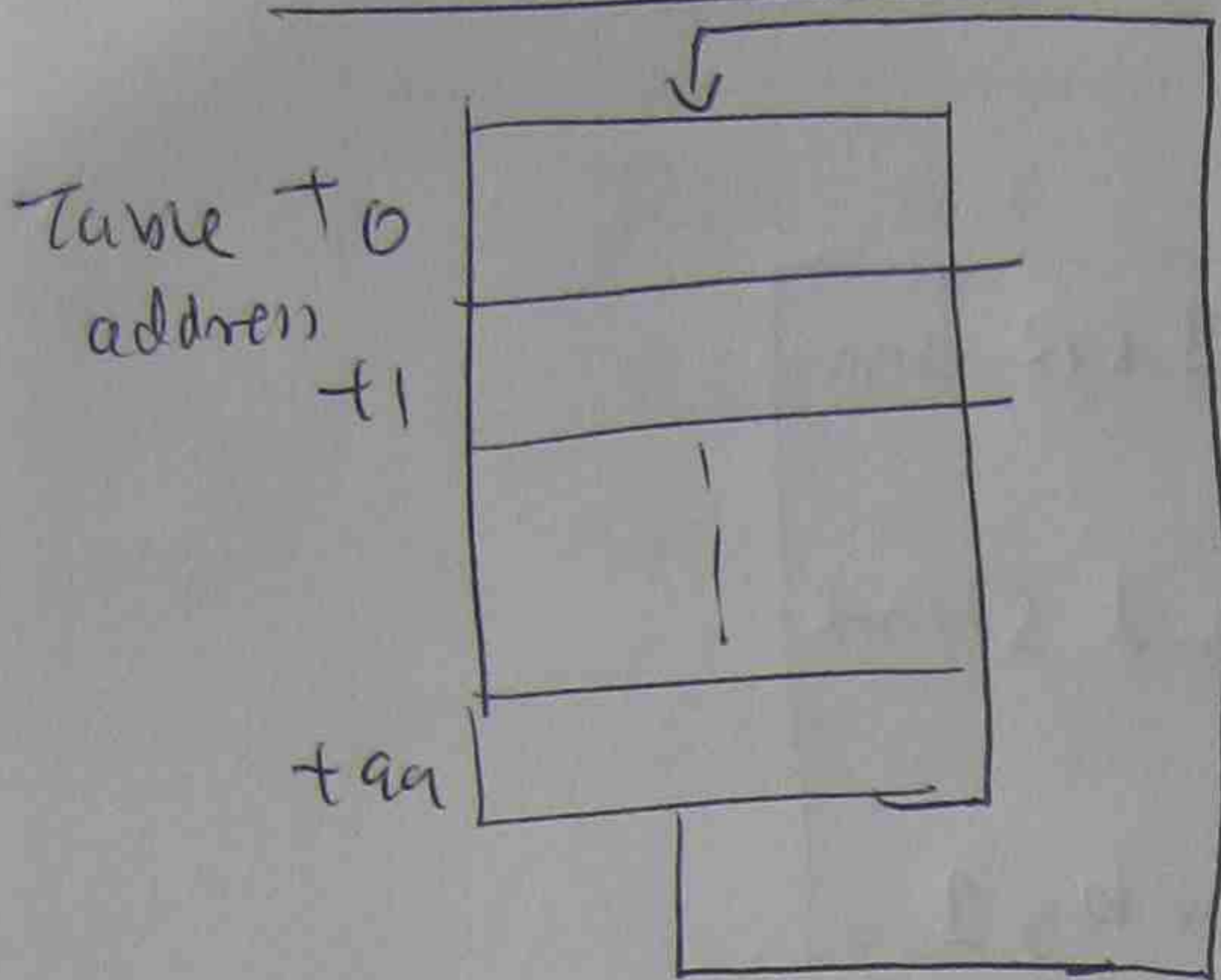
To enable a peripheral device to inform the microprocessor when it wishes to transfer data. On receipt of the interrupt, the microprocessor temporarily suspends its current activity, performs the required input or output operation, and then returns to its previous task.



Interrupt

- ① Save the contents of the program counter on the system stack
- ② Load the program counter with the start address of the interrupt service routine
- ③ Run the interrupt service routine
- ④ Finally, return control to the interrupted program by restoring the saved contents of the program counter from the system stack.

circular buffer



The main program continuously sums the values in the table together, computes the average of the last 100 values.

output this value to output port.

Table offset

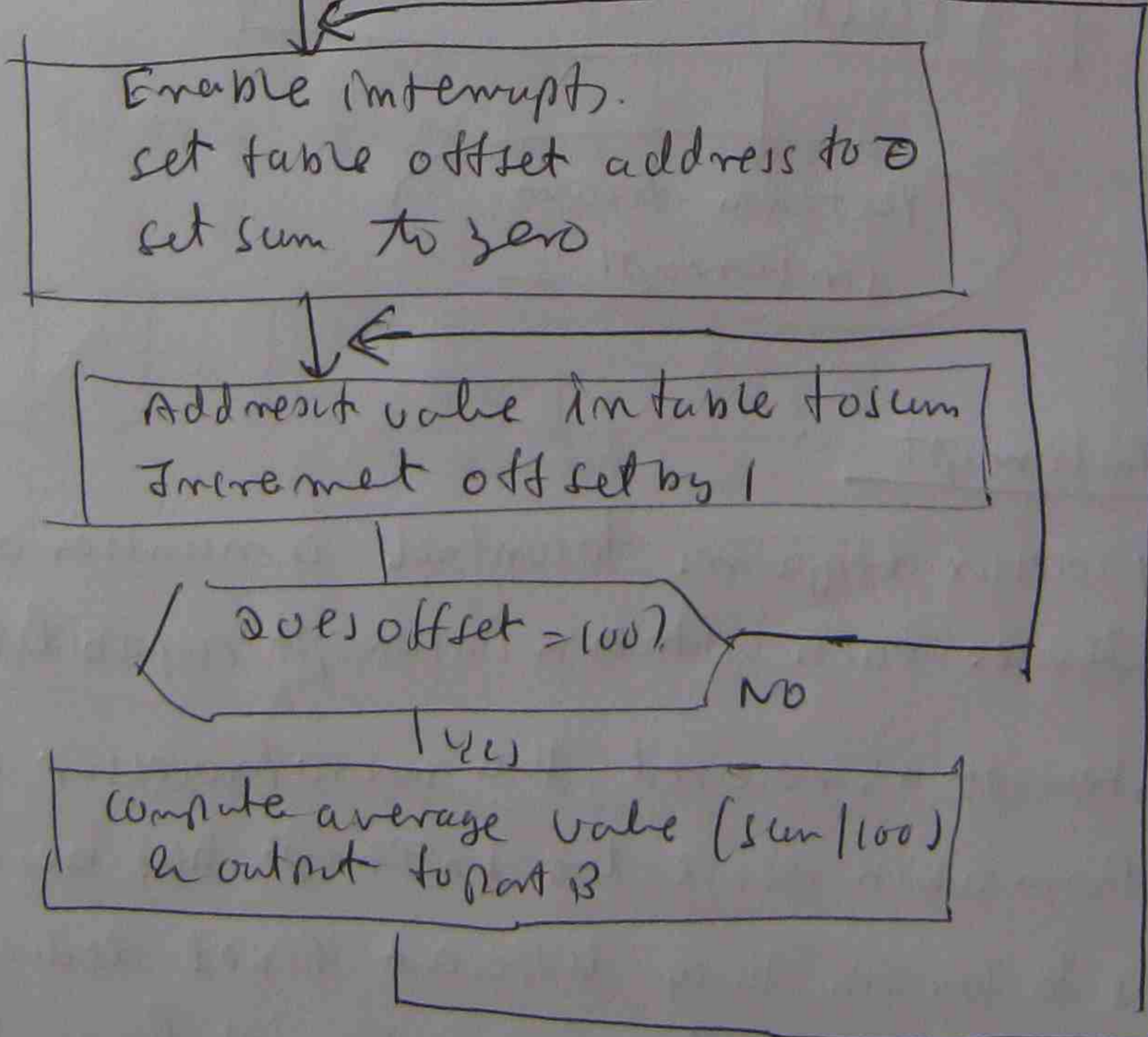
Algorithm.

BEGIN

Initialise segment

Initialise stack pointer
 Initialise Port A as an input port
 and Port B as an output port
 Clear table contents to zero
 Reset vector interrupt mask

main program segment



Q.1
84

BEAIN ISR

Interrupt
service
routine
segment

Save registers used in ISR on Stack
Read value from Port A and store in Table
Increment Table pointer by 1

Does table pointer = 100

Yes

Reset Table pointer to 0

Restore registers from Stack

Return from Interrupt

Multiple Interrupt

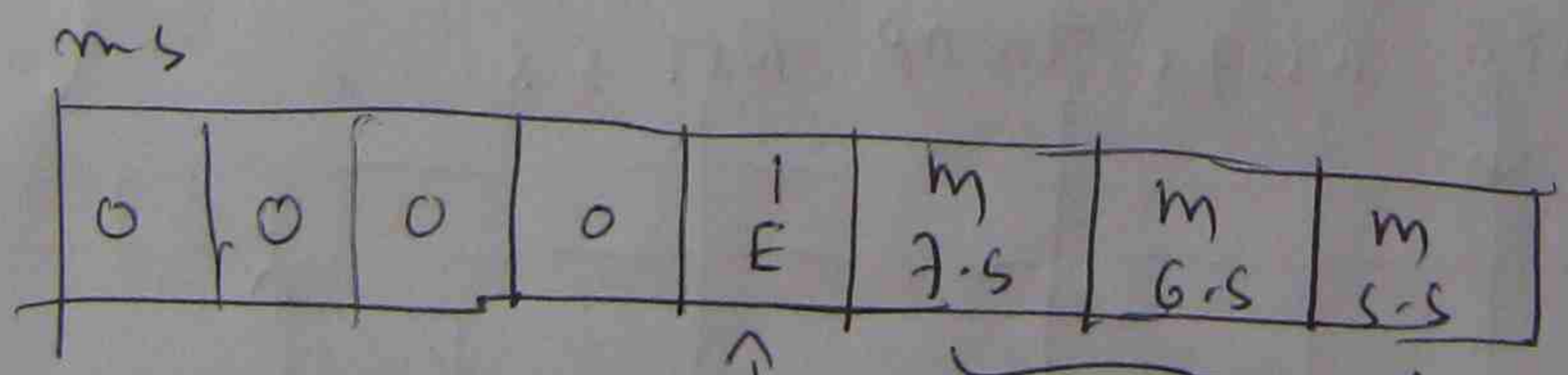
The micro processor requires to control a number of input (or) output devices each with an interrupt capability.

When an interrupt is received, the micro processor can automatically determine from which device the interrupt has been sent since it is caused to branch to a different fixed dedicated location in memory for each of the five interrupt lines. Dedicated location - vector address.

Interrupt Input	Vector address
RST 4.5 (TRAP)	0024 (hex)
RST 5.5	002C
RST 6.5	0034
RST 7.5	003C
INTR	The vector address for this input is not fixed and is part of the instruction placed on the data bus by the interrupting device when the interrupt request is acknowledged

RST 4.5 (TRAP) Highest Priority
 RST 7.5
 RST 6.5
 RST 5.5
 Lowest Priority

EI - Enable Interrupt, DI - Disable Interrupt



I-M Register format

Interrupt mask
 Interrupt Enable flag

SIM - Set Interrupt mask

The A register is first loaded with the required mask bits in the three least significant bit positions together with a logical 1 in the 4th bit and the SIM instruction is then given

} mvi A, 02
SIM

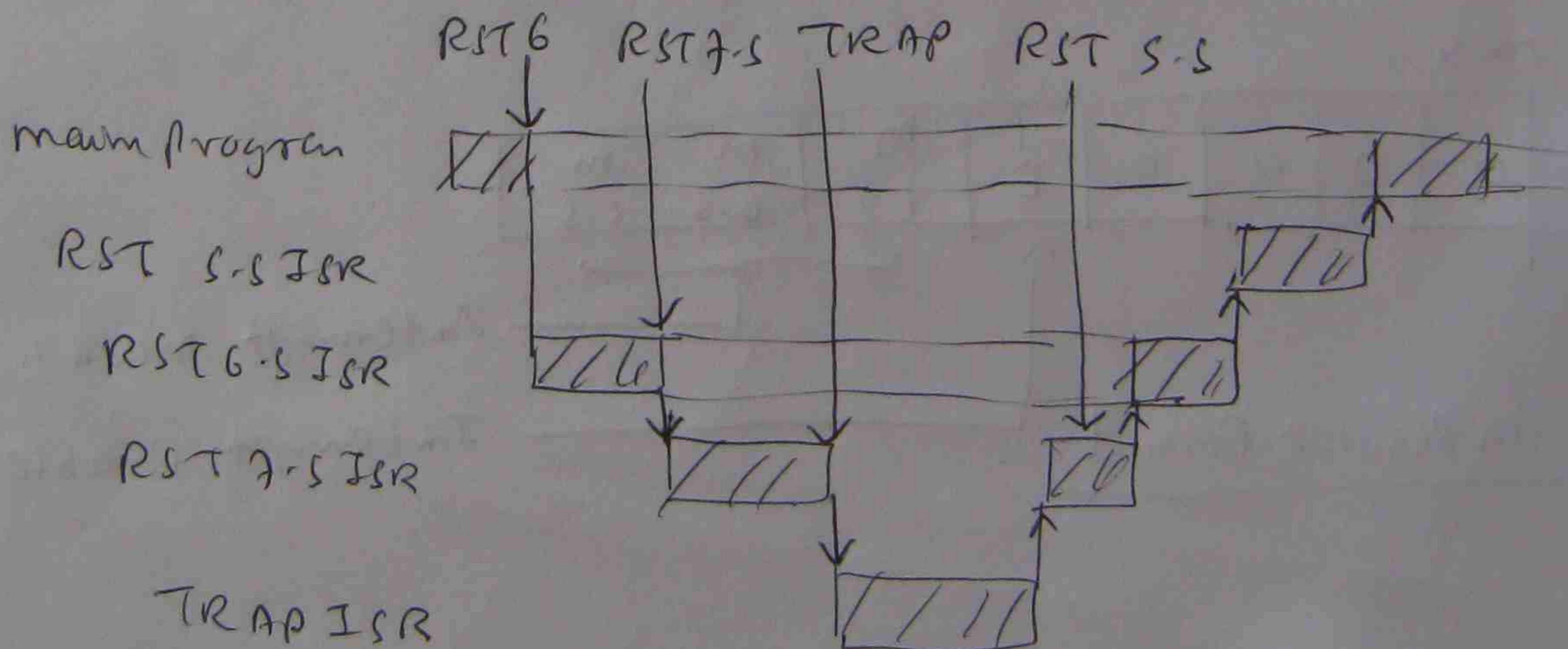
RIM - Read Interrupt mask



} After a RIM instruction the least significant 3 bits of A register indicate the state of the corresponding mask bit

Interrupt Priority Levels

Priority assignment of different interrupt inputs.

- | | | |
|---------|--------------------------------|---------------------------|
| TRAP | Power failure | Priority
HI
↓
LO |
| RST 7.5 | Over temperature alarm | |
| RST 6.5 | Real time clock | |
| RST 5.5 | Read new temperature set value | |



 Running
 Suspended

Assembly Instructions

Comments

Main Program

```

BEALM MVI A, 0B
      SIM
      EI
  
```

Reset all mask bits and enable interrupts

S-S Interrupt Service Routine

```

ISR S-S PUSH PSW
      MVI A, 00
      SIM
      EI
      RET
  
```

Disable S-S Interrupt & enable G-S & 7-S Interrupts.

G-S Interrupt Service Routine

```

ISR G-S PUSH PSW
      MVI A, 0B
      SIM
      EI
      RET
  
```

Disable S-S and G-S Interrupt and enable 7-S Interrupts.

7-S Interrupt Service Routine

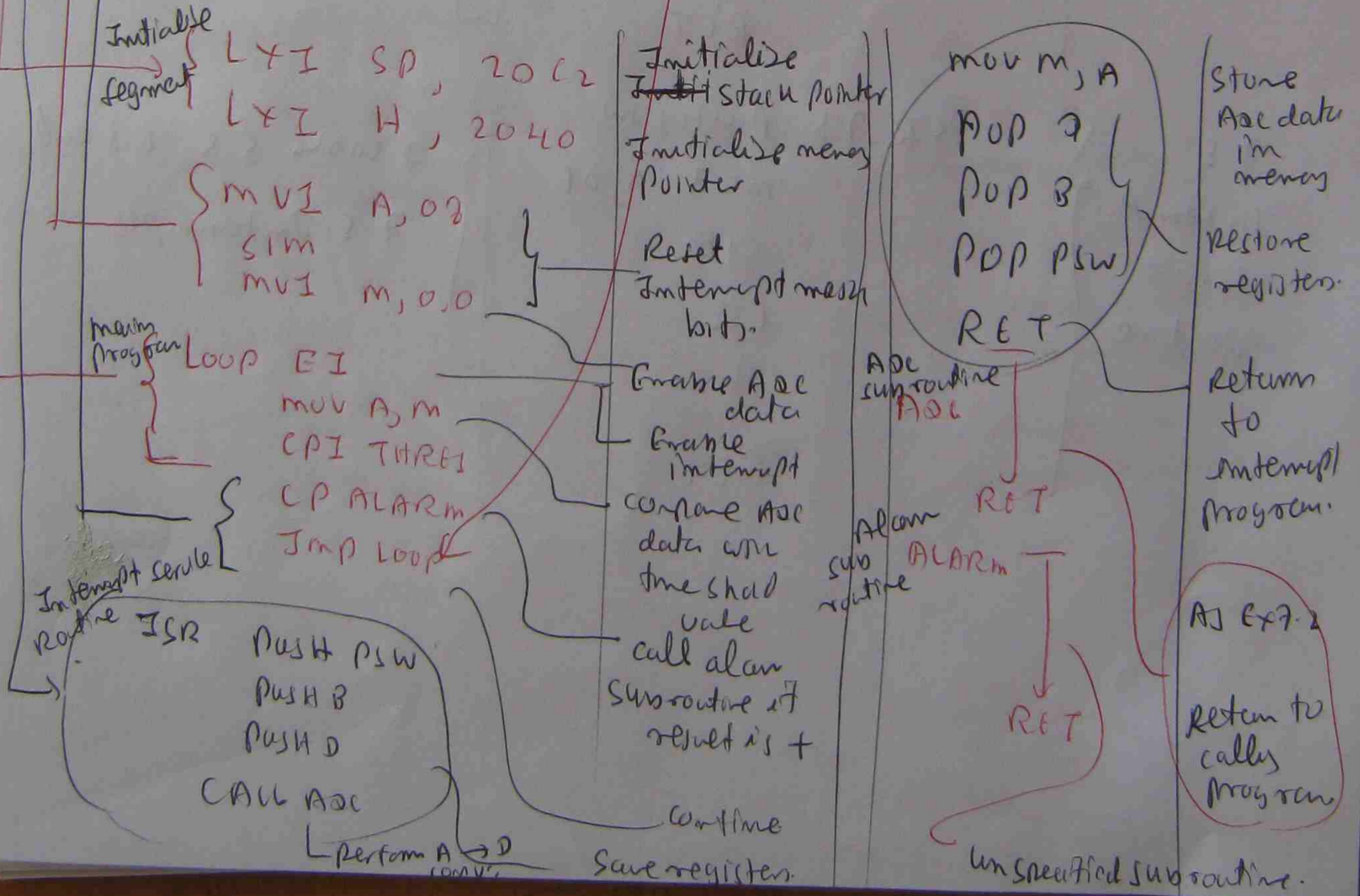
```

ISR 7-S PUSH PSW
      MVI A, 0F
      SIM
      EI
      RET
  
```

Disable S-S, G-S and 7-S Interrupts.

Use of micro processor with interrupt

- Ex 7.3
- 1 The program first initialize an area of memory as a stack
 - 2 Reset the interrupt mask bits.
 - 3 Then continuously compare the data returned by Subroutine ADC. with a preset threshold value. (THRESH)
 - 4 If this value is equalled or exceeded, Subroutine alarm is called.
 - 5 Analogue to digital conversion is performed on receipt of interrupt on RST.
 - 6 The instruction Jump ISR must therefore be stored in memory starting at the interrupt vector address 003C (hex)

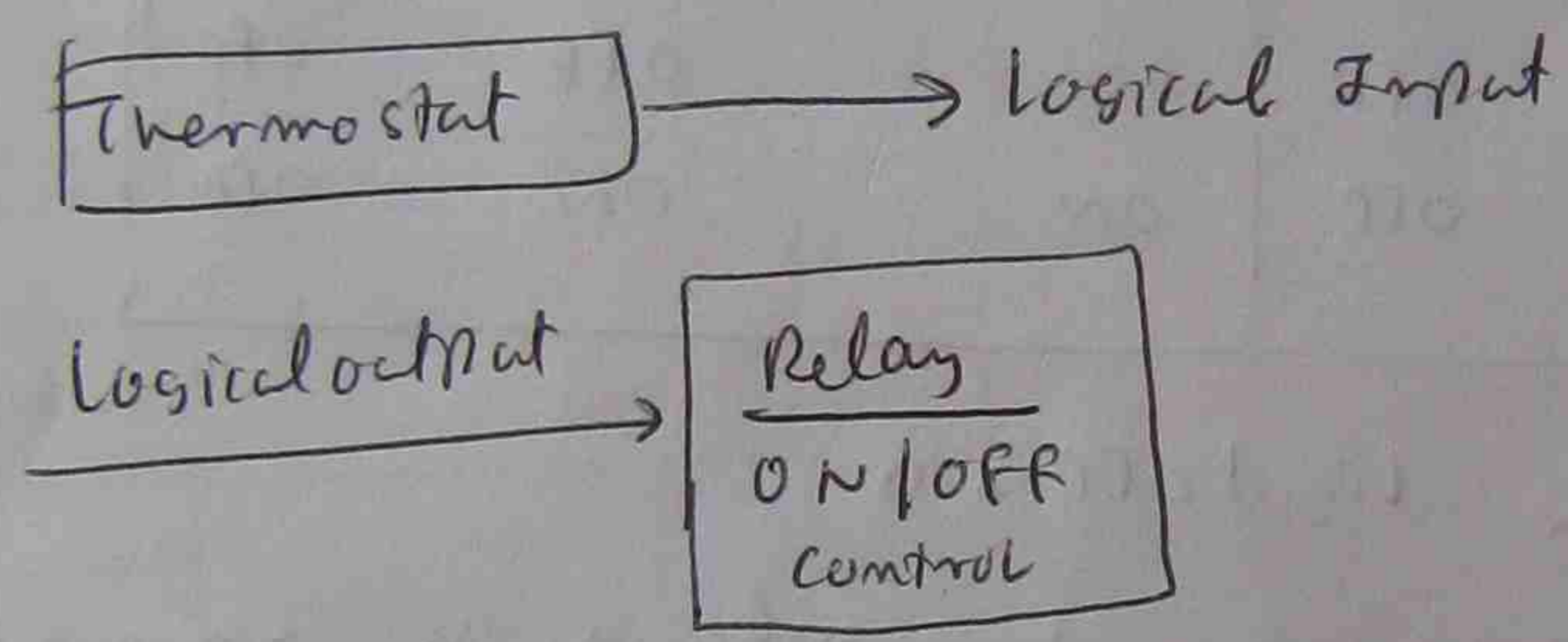


Application Examples

- Micro Processor applications
- Sequence controller
 - Digital clock, timing sequence
 - Digital to analogue converter
 - Analogue to digital converter

Basic Sequencing

- The instructions provided by a microprocessor to input & output logical data to and from an external device.
- Programmable input/output (PIO) port may be used to provide the necessary latching (2) isolation functions.



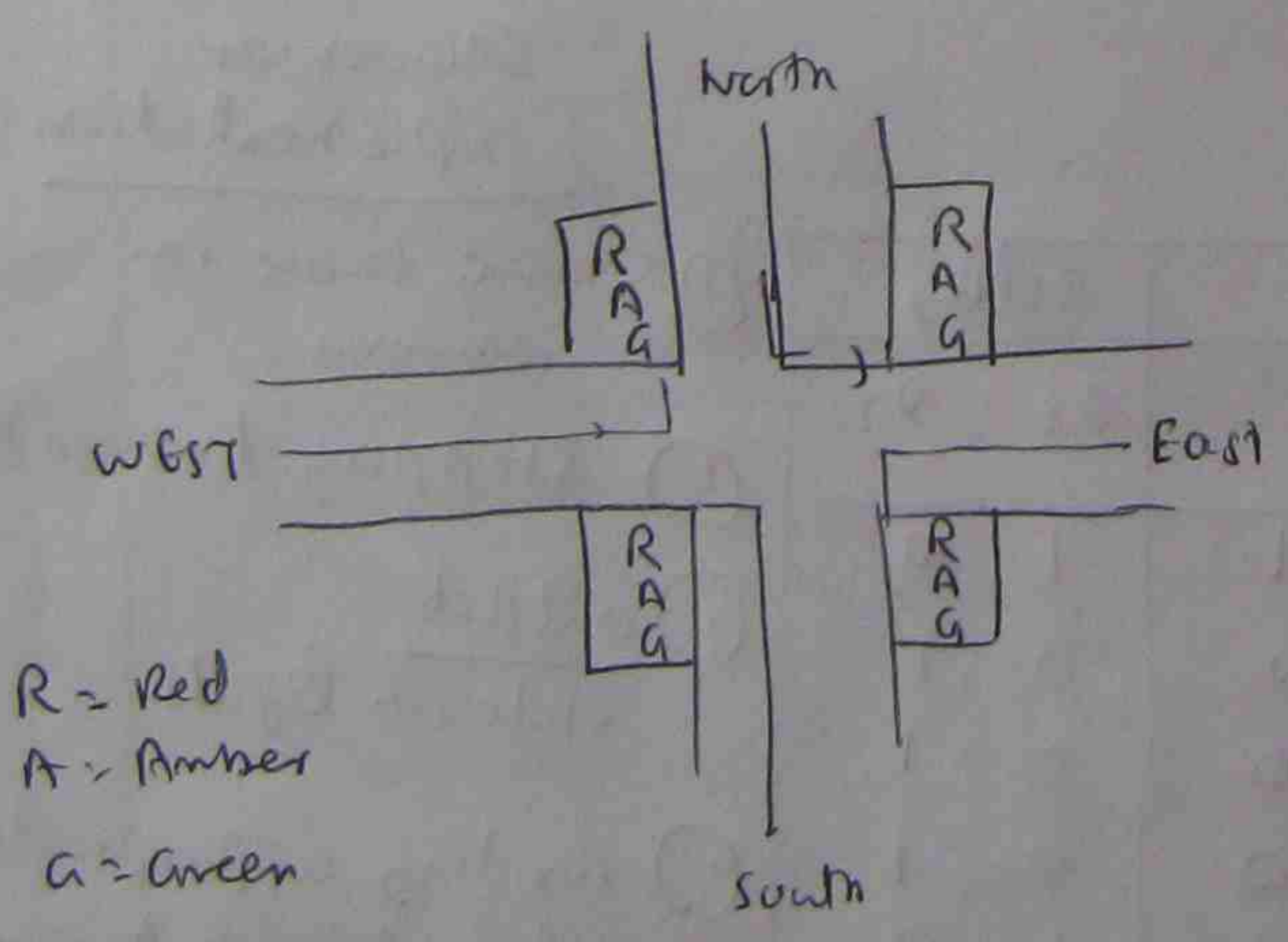
A sequencer activates a number of devices in a preset sequence with a preset time delay between each new device state.

Ex (1) Traffic Light Sequencer

I Sequencing

- Sequence

- Red, Red + Amber, Green, Amber
- Red again
- North & South - change in the same sequence
- East & West - change in the same sequence.



Traffic light sequence table

State	North / South			East / West		
	Red	Amber	Green	Red	Amber	Green
0	ON	OFF	OFF	OFF	OFF	ON
1	ON	OFF	OFF	OFF	ON	OFF
2	ON	OFF	OFF	ON	OFF	OFF
3	ON	ON	OFF	ON	OFF	OFF
4	OFF	OFF	ON	ON	OFF	OFF
5	OFF	ON	OFF	ON	OFF	OFF
6	ON	OFF	OFF	ON	OFF	OFF
7	ON	OFF	OFF	ON	ON	OFF

Light ON = 1 Light OFF = 0

Relay 1 presents → $D_1 = 1$ Relay 1 is not present $D_1 = 0$

Relay 2 presents → $D_2 = 1$ Relay 2 is not present $D_2 = 0$

Relay 1 = Long delay between overall direction changes (~ 2 min)

Relay 2 = Short delay between transitional light settings (~ 3 sec)

Traffic light state table

State	N/S			E/W			Relay	
	R	A	G	R	A	G	D_1	D_2
0	1	0	0	0	0	1	1	0
1	1	0	0	0	1	0	0	1
2	1	0	0	1	0	0	0	1
3	1	1	0	1	0	0	0	1
4	0	0	1	1	0	0	1	0
5	0	1	0	1	0	0	0	1
6	1	0	0	1	0	0	0	1
7	1	0	0	1	1	0	0	1

Sequencer Implementation

- 1) Store table in memory
- 2) Stepping through it
- 3) output state of lights.
- 4) writing appropriate delay time between steps

BISS PIO

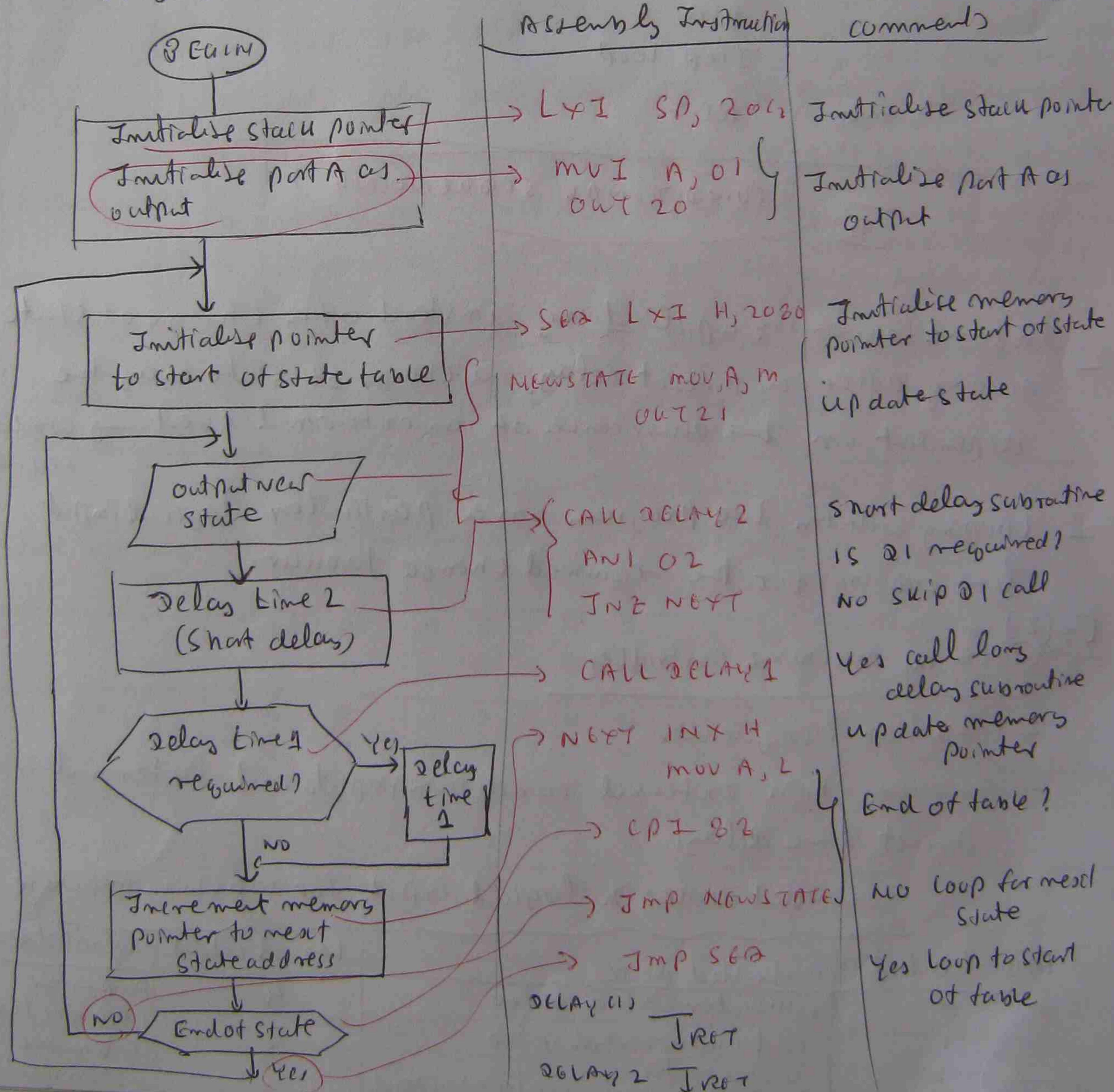
- most significant 3 bits of port A drive north/south lights (R A A)

- The next most significant 3 bits drive the East/West lights (R', A', G')

Delay time

Stage 1 - New state, a delay of θ_2 is executed

Stage 2 - if required a further delay of $\theta_1 - \theta_2$ is executed



DELAY (1)/(2)

```

TIM DLY      MVI A, 00
LOOP        CMP C
           JZ TIME

           NOP
           NOP
           NOP
           NOP
           NOP
           DCR C
           JMP LOOP

TIME        RET

```

II CONDITIONAL SEQUENCING

In some sequencing applications, instead of a change of state occurring after a preset delay, a change of state may be dependent on the occurrence of an external event. → conditional sequencing

- Looping within the program on a particular logical input line waiting for the required change to occur.

EX(2)
Washing machine controller

- conditional sequencer
- Involves both external condition inputs and internal preset time delays.

controlled devices & logical inputs for washing machine

Logical output	controlled device	Logical output	controlled device
0	Hot water control valve	4	Pump motor (empties tub)
1	cold water control valve	5	Tub motor spin speed
2	water heater		
3	tub motor / wash / rinse speed		

BEGIN

Initialize stack pointer
 Initialize PI/O ports
 Initialize pointer to start of state table

```
START LXI SP, 2002
      MVI A, 01
      OUT 20
      LXI H, 2080
```

Initialize stack pointer
 Initialize port A to be output & port B to be input
 Initialize memory pointer to start of state table

output new state

```
NEWSTATE MOV A, M
          OUT 21
```

output new state

Is time delay required?

```
ANI 03H
JZ COND
ANI 02H
JZ D2
```

Is delay required?
 No skip to conditional section
 Yes is it D1?

Read Input conditions

D1?

```
CALL DELAY1
  Imp INCM
```

Yes compute D1 skip conditional section

wait D1

wait D2

```
D2 CALL DELAY2
  Imp INCM
```

No then D2 skip conditional section

Has required condition been satisfied?

```
COND INX H
      COND1 IN 22
      SUB M
      JNZ COND1
```

Increment state table pointer
 Read conditions
 Test conditions.
 Loop until conditions are met

Increment memory pointer to next state address

```
INCM INX H
      MOV A, L
      CPI 06
```

Point to next state?
 End of sequence?

End of state?

```
JNZ NEWSTATE
```

No, loop for next state

```
HALT
```

Yes finish

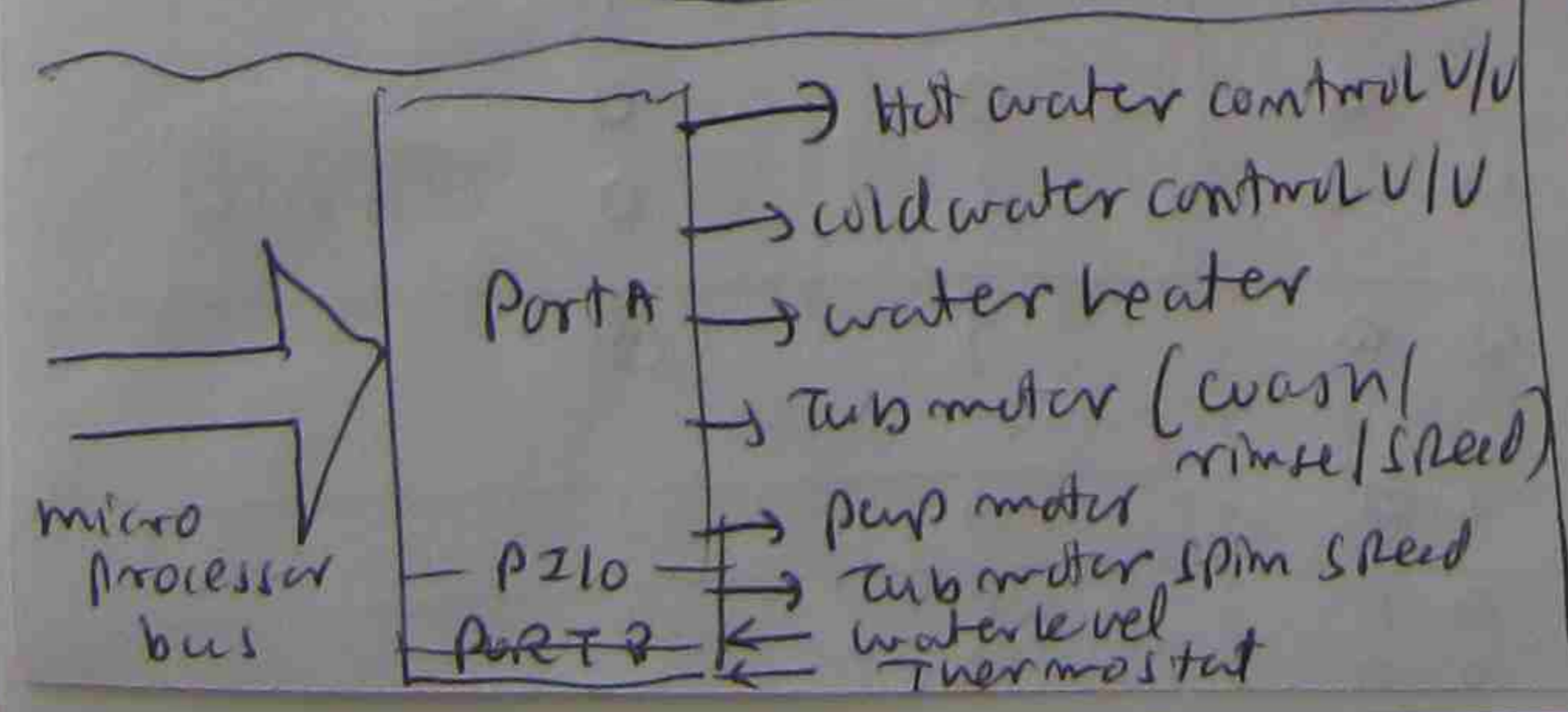
End

```
DELAY1
      RETI
```

Subroutine implementing program delay time D1

```
DELAY2
      RET
```

Subroutine implementing time D2



Programmable Timer

Intell 8155 - Single integrated circuit which incorporates a programmable timer with a 256×8 bit state RAM and 3 programmable input/output ports

+ 14 bit counter - programmed to generate either a square wave of a selectable period (or) terminal count pulse.

Timer ≤ 8.1915 ms

Timer may be programmed either to stop after the terminal count pulse is generated (OR) to continue counting and hence generate a new count pulse every, say, 8.1915 ms.

Digital clock

Exp (3)

Use the programmable timer to continuously generate a count pulse every 5 ms and to connect the timer output to one of the interrupt lines (RST 7.5) of the microprocessor.

COUNT - contains the current number of interrupts since the last change in SECS

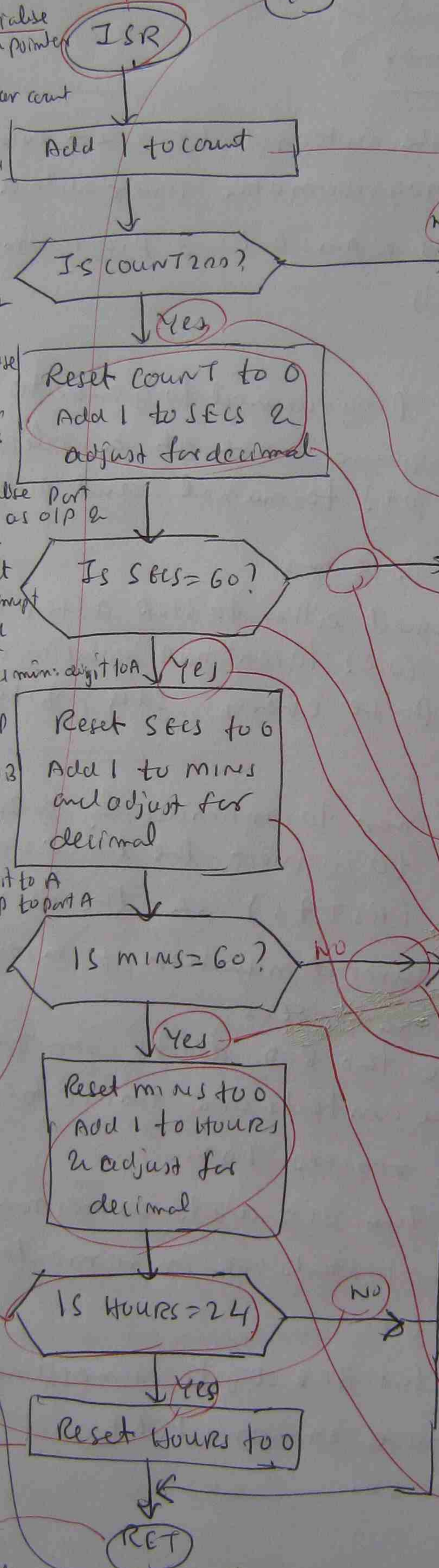
SECS - contains the two BCD digits corresponding to seconds. The contents are incremented by 1 each time COUNT reaches 200.

MINS - contains the two BCD digits corresponding to minutes. The contents are incremented by 1 each time SECS reaches 60.

HOURS - contains the two BCD digits corresponding to hours. The contents are incremented by 1 each time MINS reaches 60.

```

LXI SP, 2002
LXI H, 2000
MVI M, 00
INX H
MVI M, 00
INX H
MVI M, 00
INX H
MVI M, 12
MVI A, 10
OUT 24
MVI A, 24
OUT 25
MVI A, 03
OUT 20
MVI A, 03
SIM
Loop EI
LXI H, 2032
OUT 22
INX H
MOV A, M
OUT 21
JMP LOOP
    
```

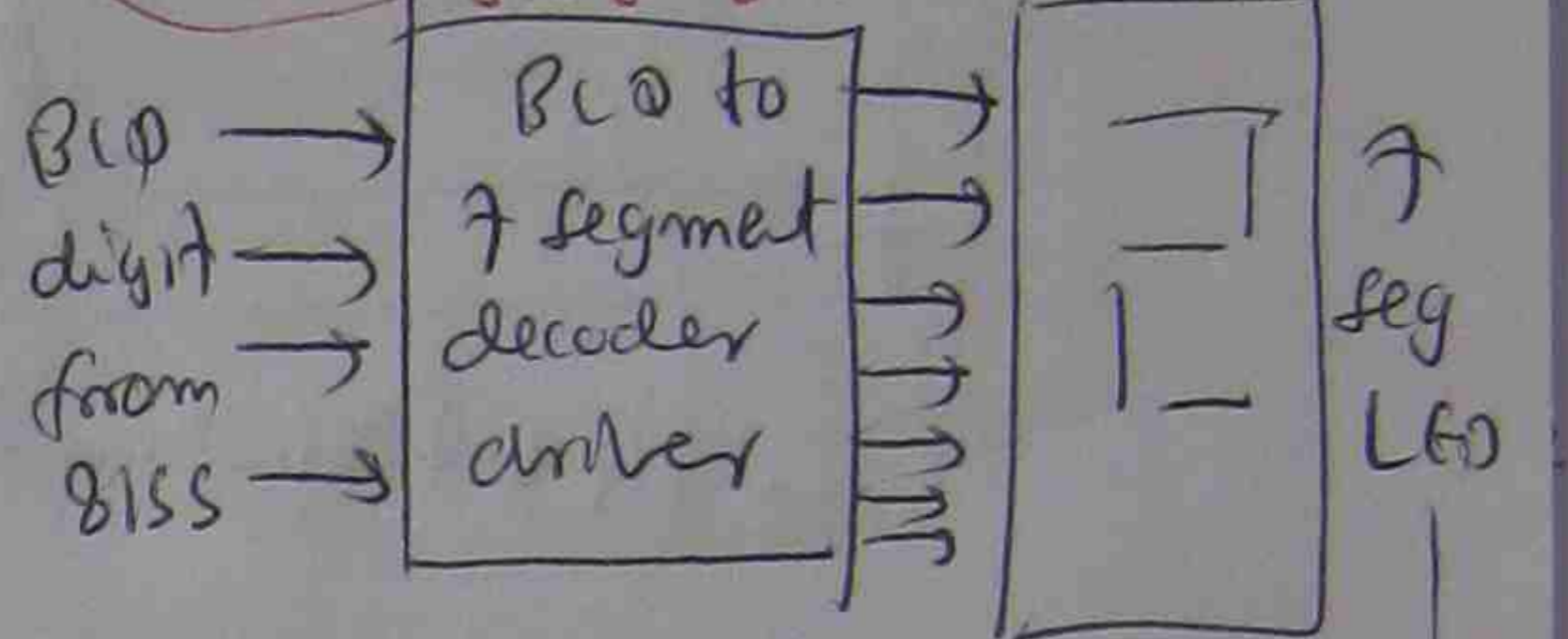


```

ISR PUSH PSW
Push H
LXI H, 2000
INR M
    
```

```

MVI A, 03
CMP M
INZ END
    
```



```

MVI M, 00
INX H
MOV A, M
INR A
RAA
MOV M, A
    
```

```

MVI A, 60
CMP M
INZ END
    
```

```

MVI M, 00
INX H
MOV A, M
INR A
RAA
MOV M, A
    
```

```

MVI A, 60
CMP M
INZ END
    
```

```

MVI M, 00
INX H
MOV A, M
INR A
RAA
MOV M, A
    
```

```

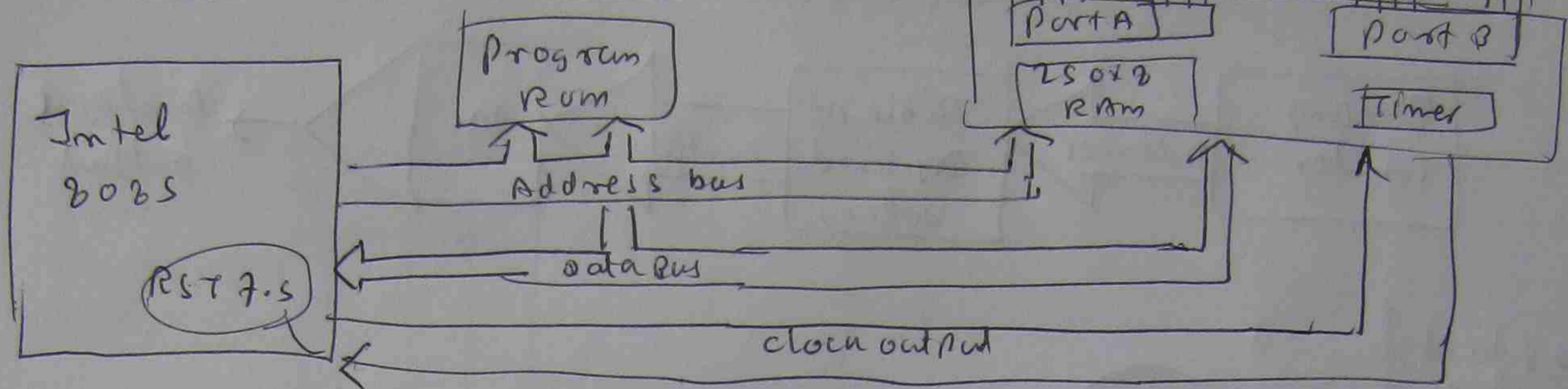
MVI A, 24
CMP M
INZ END
    
```

```

MVI M, 00
INX H
MOV A, M
INR A
RAA
MOV M, A
    
```

Return from interrupt

Digital clock schematic



RAM addresses

- 2002 - Stack pointer
- 2080 - COUNT
- 2081 - SECS
- 2082 mins
- 2083 Hours

BISS addresses

- Command Register 20 (hex)
- Port A data 21
- Port B data 22
- low order byte of count length 24
- High order byte of count length 25

IV WAVE FORM GENERATION

Direct wave form

- Use of DAC to convert the digital output from a microprocessor into an analogue form
- use of look up table.
- micro processor was used as a counter
- Its contents were incremented by unity with in a program loop
- The contents were output to DAC after each count increment.

Alternative wave form

- The contents of the counter not to drive the DAC directly
- To provide addresses to successive memory locations, the contents of which store the required digital value which is to be output to the DAC.

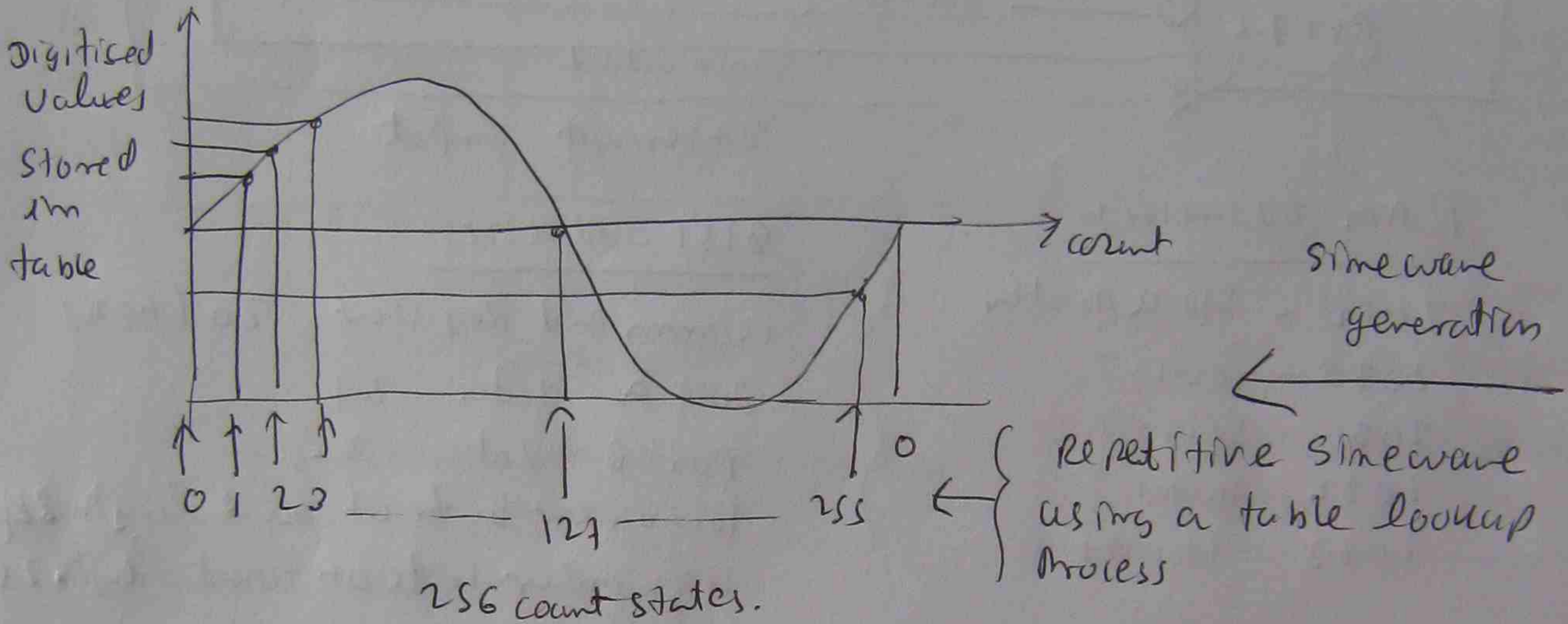
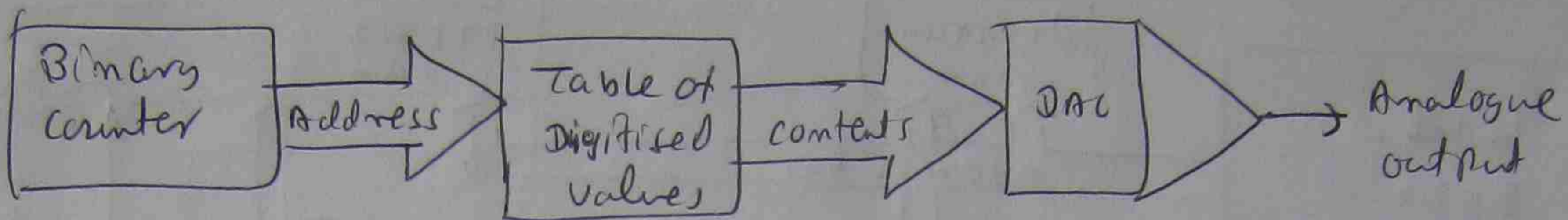
Table ← The contents of the block of successive memory locations
 ↘ Table loop up process.

DAC - 8 bits, 256 count states.

Ex 4

Wave form generation

98

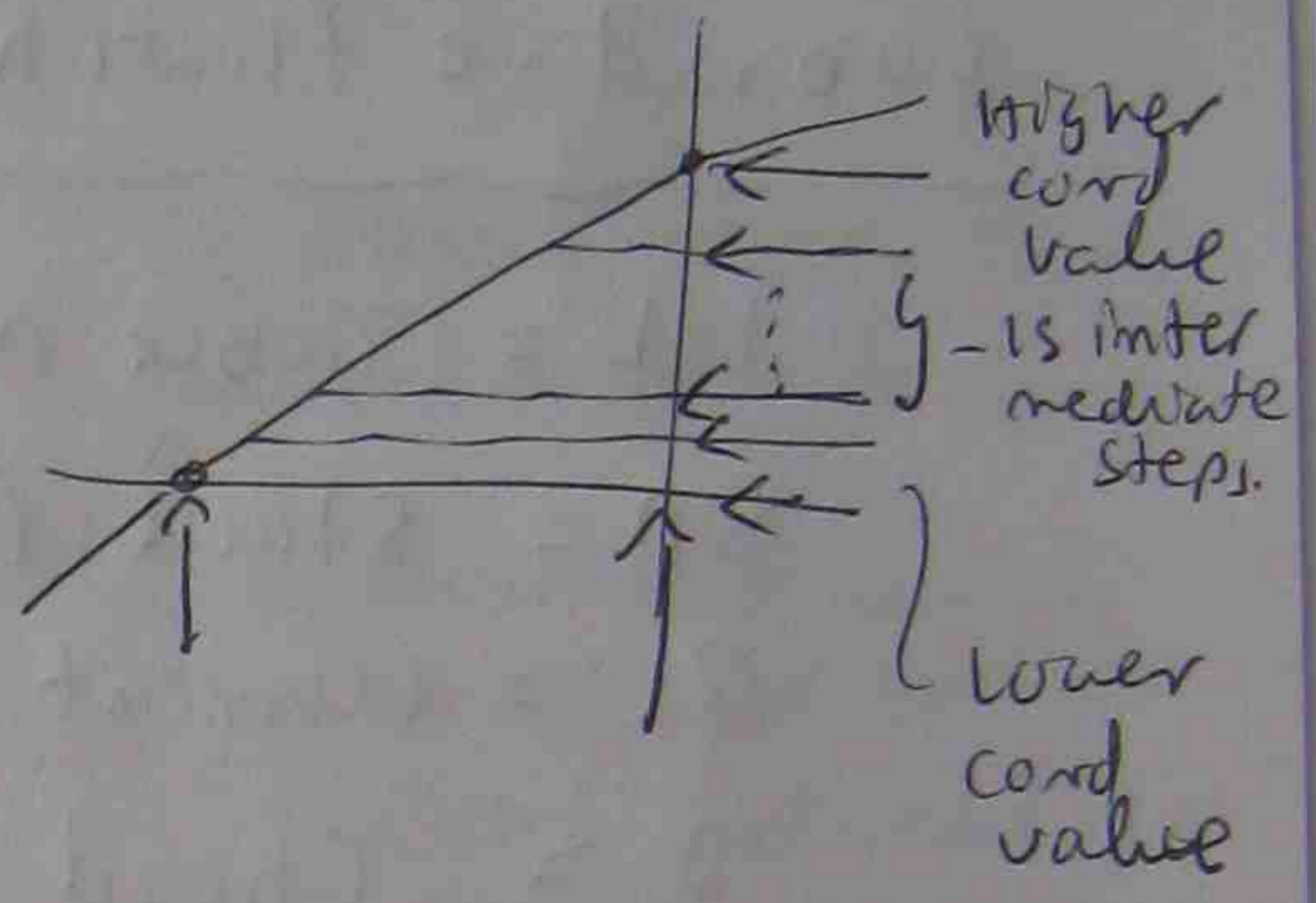
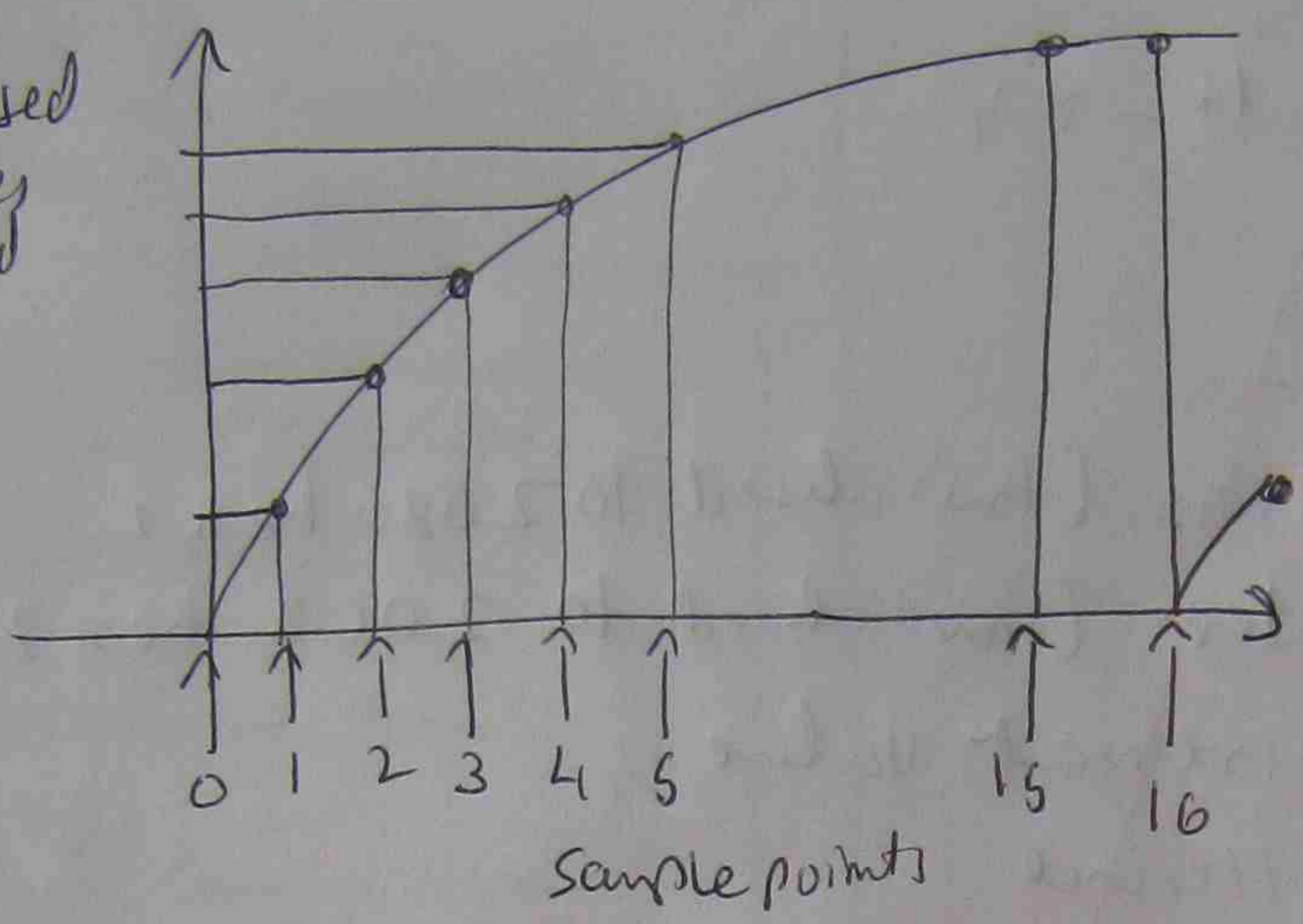


- The table is stored in memory starting at address 2000 and
- Register pair HL is initialised to this value.
- Register L is 8 bit counter & the combined contents of register pair HL provide complete 16 bit memory address automatically.
- On reaching FF, the contents of register L will return to 00
- The combined contents of HL return to 2000 & the process will repeat.

Assembly Instructions	Comments
MVI A, 01 OUT 20 → LXI H, 2000	Initialize port A as an output port Initialize register pair HL to point to start of table
→ LOOP → MOV A, M OUT 21 INR L → JMP LOOP	Loop up value from table output value to DAC Increment table offset

Piecewise Linear Interpolation

Digitised values stored in table



Piecewise Linear Interpolation

Linear interpolation process

- 1) stored values → chords
- 15 values in between → steps adjacent chords

Intermediate values between adjacent chord values are obtained by first evaluating the chord difference.

Ex 5 The program generates an exponential wave form using a table look up process and piecewise linear interpolation.

To find intermediate value, it maintains a current increment value (chord $\times n$) and simply adds the cord difference to this following each iteration.

- The current increment value is held as a 16 bit number in register pair BC.
- The combined contents are divided by 16 to form each intermediate value.
- (shifting the current increment value four places to the right. the process is performed in separate subroutine)

Subroutine

(109)

IF $(B) = 0x$ (hex) (ms 4 bits of B are always)

$(C) = yz$ (hex) (zero)

Then $(A) \leftarrow \text{result} - xy$

Subroutine flowchart

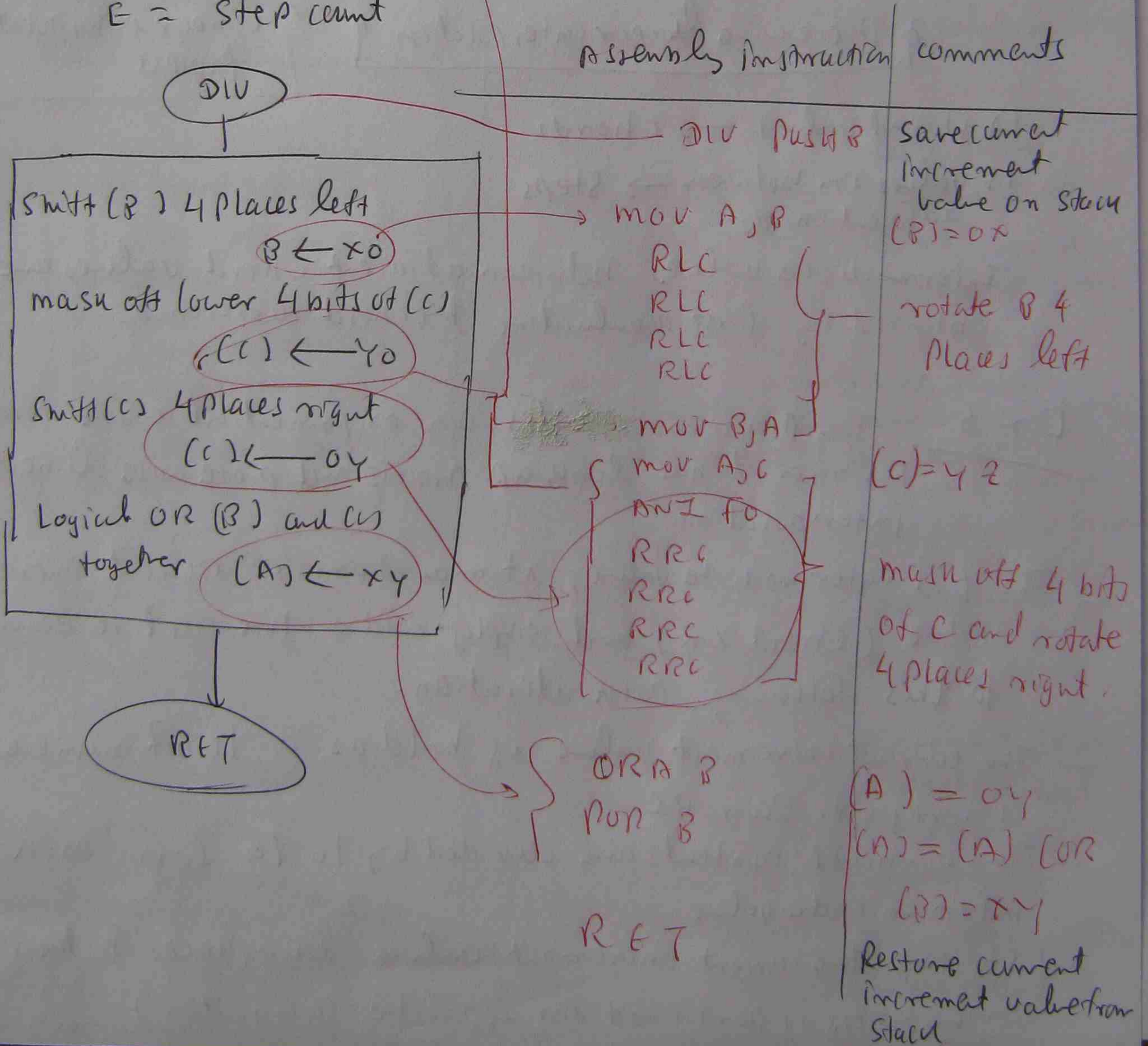
HL = Table Address (initialised to 2020 hex)

SP = Stack Pointer (initialised to 20C2 hex)

BC = Current Increment value

D = Chord Difference

E = Step count



BEA IN

Initialise Table
offset to zero

set current Increment
to zero
Read lower & higher
Chord values from
table,
compute Chord difference
Initialise stepcount
to 15

Add Chord difference
To temporary
Increment value
Divide result by 16
(subroutine DIV)
and add to lower Chord
value
output result to DAC
decrement stepcount

NO YES
Is step count = 0?

Increment Table
offset

NO YES
Is offset = 16?

```
LXI SP, 2062  
MVI A, 01  
OUT 20
```

Initialise stack
pointer
Initialise port A
as an output port

```
START LXI H, 2020  
LXI B, 0000
```

Initialise HL to
start of Table
clear Bc

```
CONT INR L  
MOV B, M  
DCR L  
SUB M  
MOV D, A
```

Obtain higher Chord
value from table
Subtract lower
Chord value to from
Chord difference &
save D

```
MVI E, 0F
```

set setupcount to 15

```
STEP MOV A, C  
ADD D  
MOV C, A  
JNC NOC  
INR B
```

Add chord
difference to
16 bit current
Increment value

```
NOC CALL DIV  
ADD M
```

Divide current
Increment value by
16 & add to lower
Chord value

```
OUT 21
```

output value to DAC

```
DCR E
```

```
JNZ STEP
```

Test if step
count = 0

```
INR L
```

```
MOV A, L
```

```
CPI 10
```

Increment Table
offset & test
if = 16

```
JNZ CONT
```

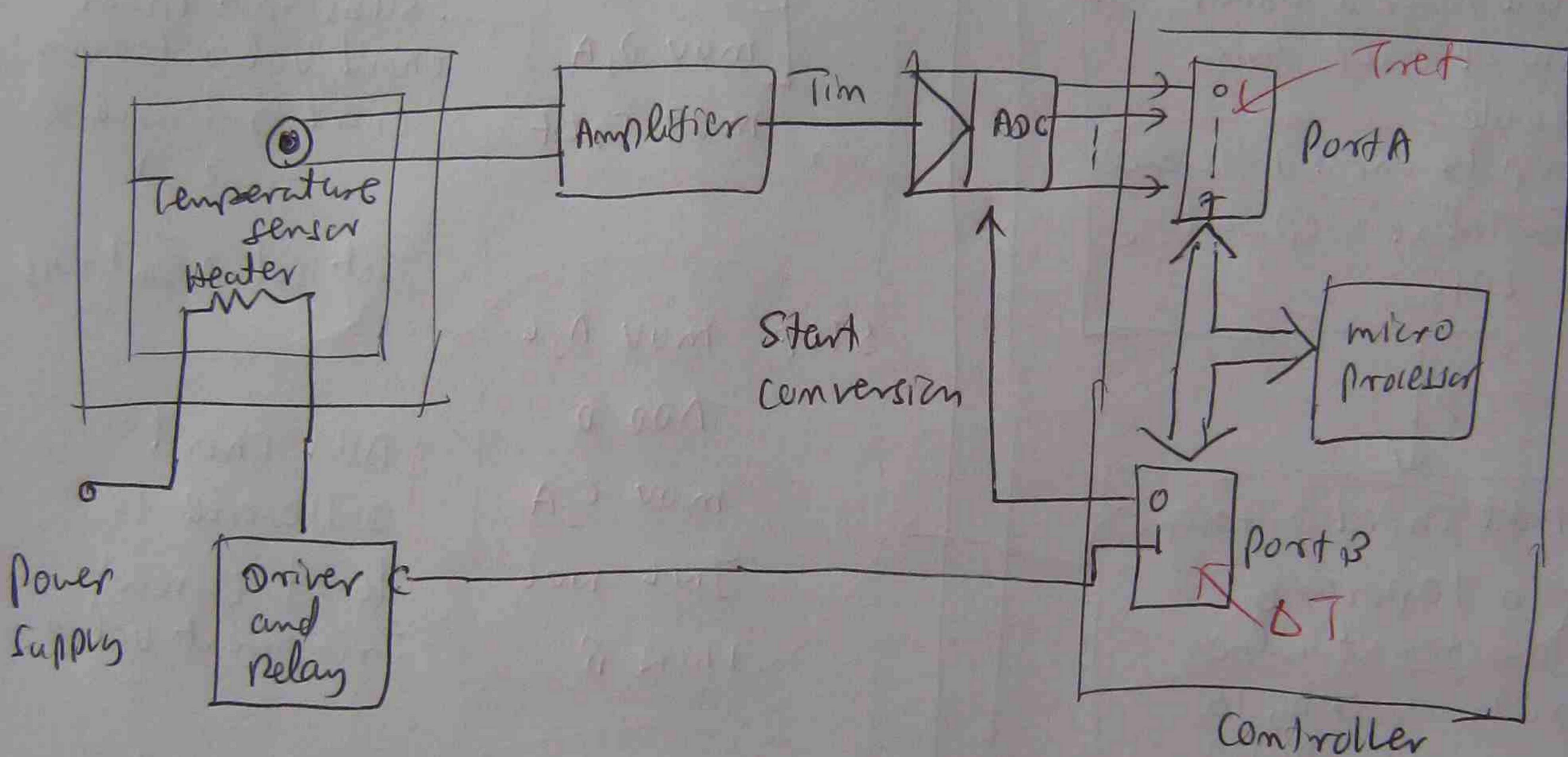
No continue

```
JMP START
```

Yes - Reinitialise
Table offset &
repeat

Temperature controller

- The required temperature T_{ref} is variable and is stored in a location in RAM.
- The tolerance limits on the required temperature $\pm \Delta T$ are also assumed variable and stored in a second RAM location.

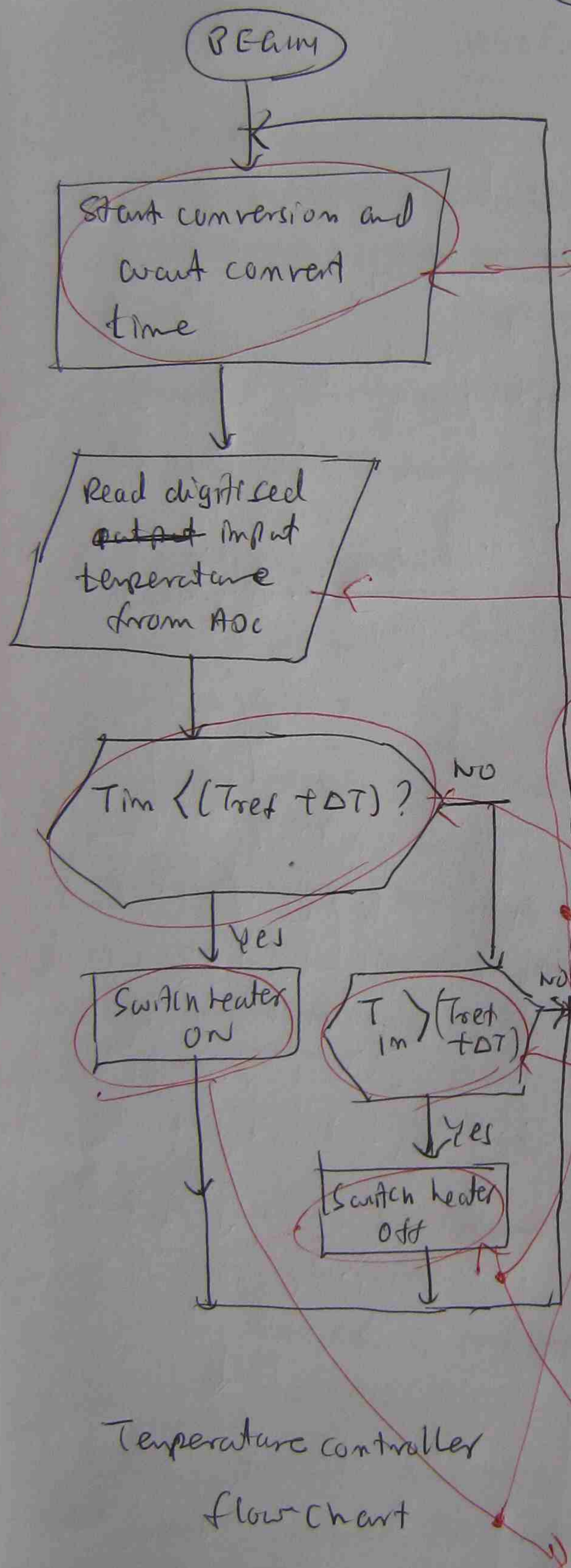


- The controlled temperature is controlled by first deriving an analogue voltage proportional to the temperature - from a thermistor & associated amplifier.
- converting this voltage into an 8 bit digital value by means of an ADC
- supply to heater is controlled by a single digital output from the micro processor via a suitable driver & relay

1 \rightarrow heater on 0 \rightarrow heater off

If Input $T_{im} > T_{ref} + \Delta T \Rightarrow$ Heater is ~~off~~ turned off

If Input $T_{im} < T_{ref} - \Delta T \Rightarrow$ Heater is turned on



```

LXI H, 2080
MOV A, M
MOV E, A
INX H
SUB M
MOV B, A
MOV A, E
ADD M
MOV C, A

MVI A, 02
OUT 20

MVI P, 00
MOV A, 0
OUT 22

START: MOV A, 0
        ORI 01
        OUT 22
        ANI 02
        OUT 22
        CALL DELAY
        IN 21
        CMP B
        JC HTON
        JZ START
        CMP C
        JL START
        JZ START

MVI A, 00
MOV P, A
OUT 22
JMP START

HTON: MVI A, 01
        MOV P, A
        OUT 22

DELAY: RET
    
```

Temperature controller flow chart

Read Tref from A2
Save in E

Form (Tref - ΔT) and store in B

Form (Tref + ΔT) and store in C

Initialize port A as input
port B as output

Switch heater off

Recall state of port B

Switch bit 0 on
Switch bit 0 off

Start ADC

wait conversion time

Read Tim from ADC

If Tim < (Tref - ΔT)
Jump to switch on (HTON)

If Tim > (Tref + ΔT)
Jump to start switch heater off

Switch heater on

Development Aids

- Single board system - Intel 8085 microprocessor microprocessor, system clock source, Random access memory for holding a system monitor program, a number of input/output ports, key pad, associated numeric display, bus control logic.
- Program stored in ROM
- development phase for program stored in RAM.

monitor command

digit displayed are in hexadecimal code.

x x x x . x x

address field

data or content field

- * Reset → Enable the user to force the monitor to restart from the beginning of the program
- Substitute memory - Allow the user to examine the contents of successive memory location
If required, to modify its contents

Run - Execute a program which is already stored in RAM.

Run key is first pressed, followed by 4 digit start address.

Examine register allows the users to display and if required, modify the contents of each of the microprocessor registers.

Single Step - Enable the user to examine the state of the complete system.

To overcome the errors, additional software / hardware are provided -

Assembly Language → Assembler → machine code

High level ~~machine code~~ language → Compiler → machine code.

Assembler

LDA SECS Increment contents of memory

INR A Location with symbolic name SECS

STA SECS by unit

SECS DS 1 define one storage location for SECS

LXI A SECS This load the absolute value in to register pair AL

INR m contents of memory location
SECS (addresses) are incremented by 1

SECS EQU 2000H define SECS to be absolute value 2000 (hex)

macro

SHIFT : MACRO start of macro SHIFT

RR0

RR0

RR0

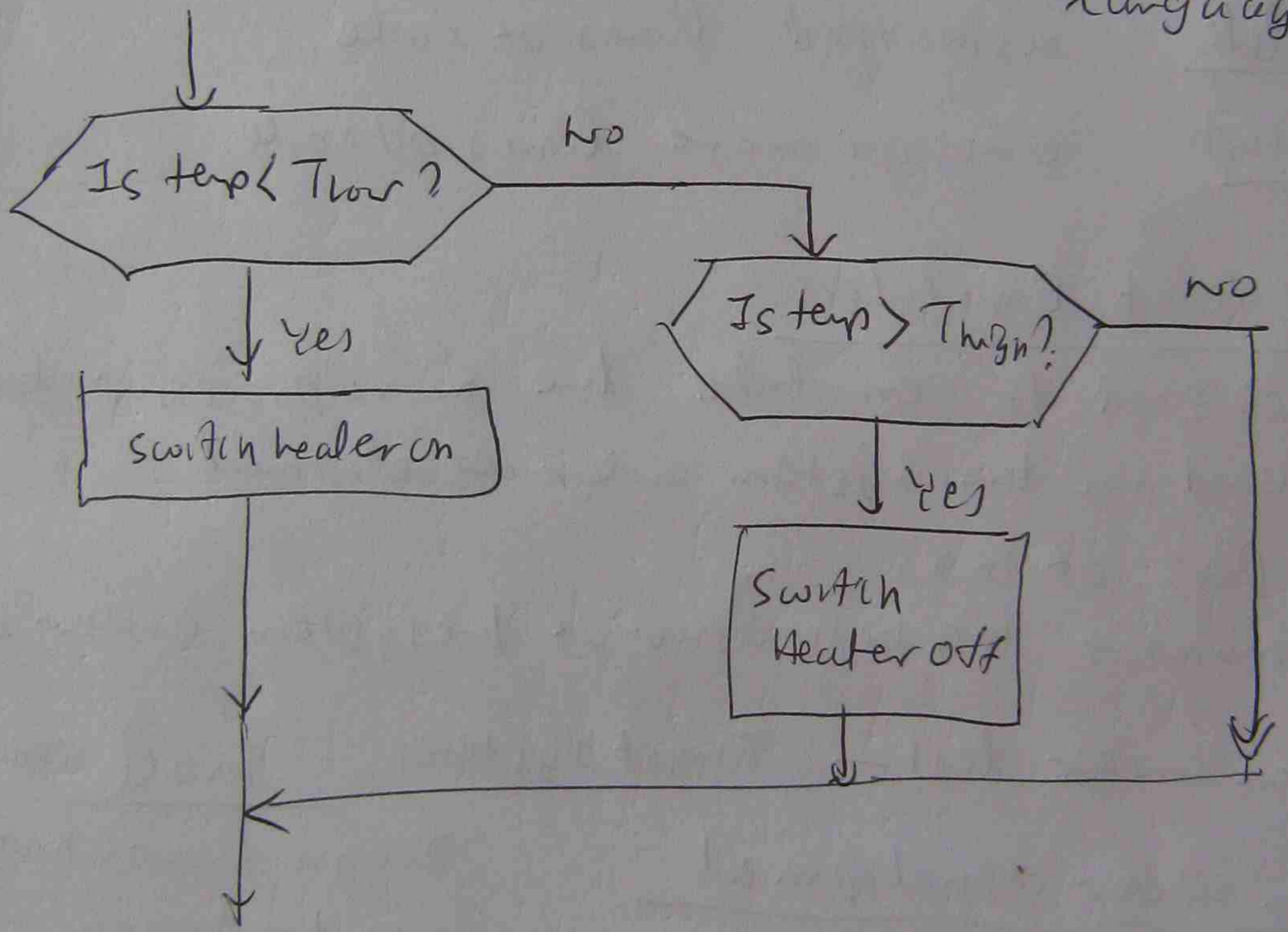
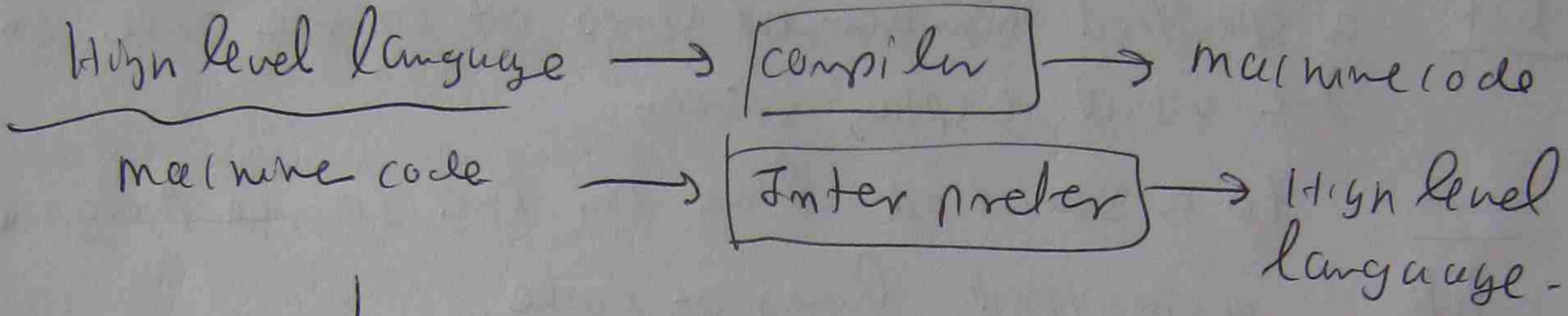
RR0

MEND

end of macro

To define the start address of compiler

ORG 2000H
List of program
Instructions
END



=====
=====
=====

```

IF Temp < Tlow THEN HEATER :- 1
ELSE IF Temp > Thigh THEN HEATER :- 0
  
```

Subroutine CALL DELAY (COUNT)

Editors

Enable the user to readily modify the source program code
Run interactively.

list a specified number of lines of source code on the visual display screen

move to a specified line in the source program

delete specified lines of code

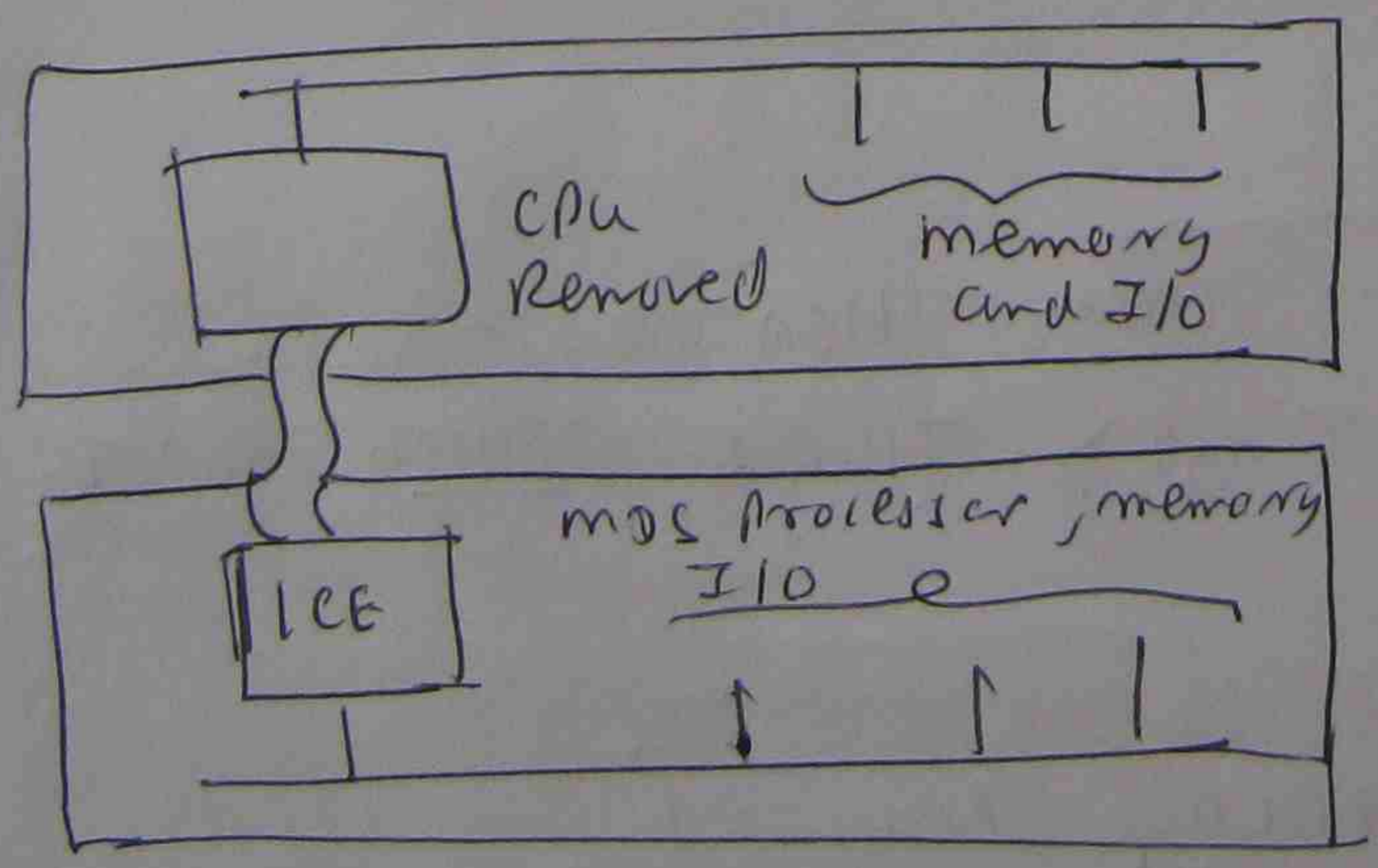
insert one or more lines of code.

In Circuit Emulators

- designed to emulate the microprocessor being used in the system under development
- Part of mas.
- monitor the behaviour of the system under development

System under test - Target system MOS (Development System)

System under development



In circuit emulation

- Design flow chart
- code the program
- Enter the program into MOS memory
- Assemble / compile
- Run the object program
- debug using ICE
- Edit the source program

Proportional-Integral-Derivative Control

Dr M.J. Willis

Dept. of Chemical and Process Engineering
University of Newcastle

e-mail: mark.willis@ncl.ac.uk

Written: 17th November, 1998
Updated: 6th October, 1999

Aims and Objectives

The PID algorithm is the most popular feedback controller used within the process industries. It has been successfully used for over 50 years. It is a robust easily understood algorithm that can provide excellent control performance despite the varied dynamic characteristics of process plant. These lecture notes,

- introduce the Proportional- Integral- Derivative (PID) control algorithm.
- discuss the role of the three modes of the algorithm.
- highlight different algorithm structures.
- Discuss methods that have evolved over the last 50 years as aids in control loop tuning.

After completion of this section of the course a student should be capable of approaching a loop tuning problem in a competent and efficient manner and have sufficient knowledge to effectively tune a PID control algorithm.

The Proportional-Integral-Derivative (PID) algorithm

As the name suggests, the PID algorithm consists of three basic modes, the Proportional mode, the Integral and the Derivative modes. When utilising this algorithm it is necessary to decide which modes are to be used (P, I or D ?) and then specify the parameters (or settings) for each mode used. Generally, three basic algorithms are used P, PI or PID.

A Proportional algorithm

The mathematical representation is,

$$\frac{mv(s)}{e(s)} = k_c \text{ (Laplace domain) or } mv(t) = mv_{ss} + k_c e(t) \text{ (time domain)} \quad (3)$$

The proportional mode adjusts the output signal in direct proportion to the controller input (which is the error signal, e). The adjustable parameter to be specified is the controller gain, k_c . *This is not to be confused with the process gain, k_p .* The larger k_c the more the controller output will change for a given error. For instance, with a gain of 1 an error of 10% of scale will change the controller output by 10% of scale. Many instrument manufacturers use Proportional Band (PB) instead of k_c .¹

The time domain expression also indicates that the controller requires calibration around the steady-state operating point. This is indicated by the constant term mv_{ss} . This represents the 'steady-state' signal for the mv and is used to ensure that at zero error the cv is at setpoint. In the Laplace domain this term disappears, because of the 'deviation variable' representation.

A proportional controller reduces error but does not eliminate it (unless the process has naturally integrating properties), i.e. an offset between the actual and desired value will normally exist.

A proportional integral algorithm

The mathematical representation is,

$$\frac{mv(s)}{e(s)} = k_c \left[1 + \frac{1}{T_i s} \right] \text{ or } mv(t) = mv_{ss} + k_c \left[e(t) + \frac{1}{T_i} \int e(t) dt \right] \quad (4)$$

The additional integral mode (often referred to as reset) corrects for any offset (error) that may occur between the desired value (setpoint) and the process

¹ This is defined as the range over which the error must change in order to drive the controller output over full range. The PB also tells you how large the error has to be before the manipulated variable reaches 0 or 100%. The PB is generally centered around the setpoint causing the output to be at 50% when the setpoint and the process output are equal.

output automatically over time². The adjustable parameter to be specified is the integral time (Ti) of the controller.

Where does the term reset come from?

Reset is often used to describe the integral mode. Reset is the time it takes for the integral action to produce the same change in mv as the P modes initial (static) change. Consider the following figure,

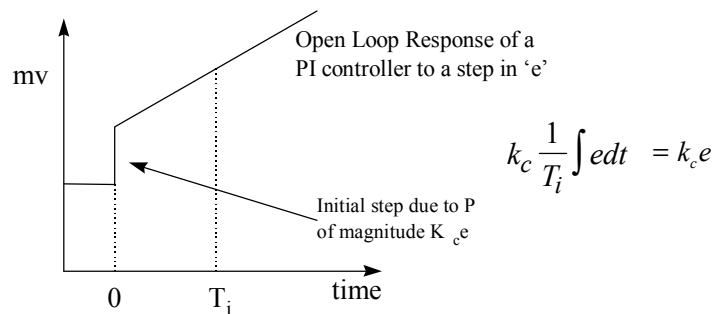


Figure (1) The response of a PI algorithm to a step in error

Figure (1) shows the output that would be obtained from a PI controller given a step change in error. The output immediately steps due to the P mode. The magnitude of the step up is $K_c e$. The integral mode then causes the mv to 'ramp'. Over the period 'time 0 to time T_i ' the mv again increases by $K_c e$.

Integral wind-up

When a controller that possesses integral action receives an error signal for significant periods of time the integral term of the controller will increase at a rate governed by the integral time of the controller. This will eventually cause the manipulated variable to reach 100 % (or 0 %) of its scale, i.e. its maximum or minimum limits. This is known as integral wind-up. A sustained error can occur due to a number of scenarios, one of the more common being control system 'override'. Override occurs when another controller takes over control of a particular loop, e.g. because of safety reasons. The original controller is not switched off, so it still receives an error signal, which through time, 'winds-up' the integral component unless something is done to stop this occurring. There are many techniques that may be used to stop this

² Different control manufacturers use different definitions for the integral mode of a controller. It can be defined as minutes, minutes/repeat or repeats per minute. The difference is very important to note so as to ensure problems do not occur during a tuning exercise. Remember the 'name game'. T_i is the integral time (minutes), if specified as repeats / minute then it is $1/T_i$ that must be entered into the controller, while minutes / repeat is again T_i . This is confusing and is compounded by the fact that manufacturers are not consistent !

happening. One method is known as 'external reset feedback' (Luyben, 1990). Here, the signal of the control valve is also sent to the controller. The controller possess logic that enables it to integrate the error when its signal is going to the control value, but breaks the loop if the override controller is manipulating the valve.

A Proportional Integral Derivative algorithm

The mathematical representation is,

$$\frac{mv(s)}{e(s)} = k_c \left[1 + \frac{1}{T_i s} + T_D s \right] \text{ or } mv(t) = mv_{ss} + k_c \left[e(t) + \frac{1}{T_i} \int e(t) dt + T_D \frac{de(t)}{dt} \right] \quad (5)$$

Derivative action (also called rate or pre-act) *anticipates* where the process is heading by looking at the time rate of change of the controlled variable (its derivative). T_D is the 'rate time' and this characterises the derivative action (with units of minutes). In theory derivative action should always improve dynamic response and it does in many loops. In others, however, the problem of noisy signals makes the use of derivative action undesirable (differentiating noisy signals can translate into excessive mv movement).

Derivative action depends on the slope of the error, unlike P and I. If the error is constant derivative action has no effect.

Revision Exercise

Use Matlab / Simulink to explore the effect a step change in error has on the various modes of an ideal PID control algorithm. Assume that $k_c = 1$, $T_i = 10$ mins and $T_D = 5$ mins.

PID algorithms can be different

Not all manufactures produce PID's that conform to the ideal 'textbook' structure. So before commencing tuning it is important to know the configuration of the PID algorithm! The majority of 'text-book' tuning rules are only valid for the ideal architecture. If the algorithm is different then the controller parameters suggested by a particular tuning methodology will have to be altered.

Ideal PID

The mathematical representation of this algorithm is:

$$\frac{mv(s)}{e(s)} = k_c \left[1 + \frac{1}{T_i s} + T_D s \right]$$

One disadvantage of this ideal 'textbook' configuration is that a sudden change in setpoint (and hence e) will cause the derivative term to become very large and thus provide a "derivative kick" to the final control element - this is undesirable. An alternative implementation is

$$mv(s) = k_c \left[1 + \frac{1}{T_i s} \right] e(s) + T_D scv(s)$$

The derivative mode acts on the measurement and not the error. After a change in setpoint the output will move slowly avoiding "derivative kick" after setpoint changes. This is therefore a standard feature of most commercial controllers.

Series (interacting) PID

The mathematical representation of this algorithm is:

$$\frac{mv(s)}{e(s)} = k_c \left[1 + \frac{1}{T_i s} \right] T_D s$$

As with the ideal implementation the series mode can include either derivative on the error or derivative on the measurement. In which case, the mathematical representation is,

$$\frac{mv(s)}{e(s)} = k_c \left[1 + \frac{1}{T_i s} \right] \text{ where } e(s) = SP - T_D scv(s)$$

Parallel PID

The mathematical description is,

$$mv(s) = k_c e(s) + \frac{1}{T_i s} e(s) + T_D s e(s)$$

The proportional gain only acts on the error, whereas with the ideal algorithm it acts on the integral and derivative modes as well.

Revision Exercises

1. Draw the block diagram representation of the ideal, series (interacting) and parallel PID control laws.
2. Write down the 'time-domain' mathematical representation of the ideal (without derivative kick) , series (interacting) and parallel PID control laws.

- Suppose that the controller settings for an ideal PID algorithm are given by, k_c , T_i , T_D . Work out the conversion factors required to ensure that a parallel implementation of the PID algorithm will provide the same mv signal given the same error signal.

Controller tuning

Controller tuning involves the selection of the best values of k_c , T_i and T_D (if a PID algorithm is being used). This is often a subjective procedure and is certainly process dependent. A number of methods have been proposed in the literature over the last 50 years. However, recent surveys indicate,

- 30 % of installed controllers operate in manual.
- 30 % of loops increase variability.
- 25 % of loops use default settings.
- 30 % of loops have equipment problems.

A possible explanation for this is lack of understanding of process dynamics, lack of understanding of the PID algorithm or lack of knowledge regarding effective tuning procedures. This section of the notes concentrates on PID tuning procedures. The suggestion being that if a PID can be properly tuned there is much scope to improve the operational performance of chemical process plant.

When tuning a PID algorithm, generally the aim is to match some preconceived 'ideal' response profile for the closed loop system. The following response profiles are typical.

Servo Control

For a unit step change in setpoint (0 - 1) the two response profiles shown in figure 2 could be obtained (depending upon the process dynamics and controller settings),

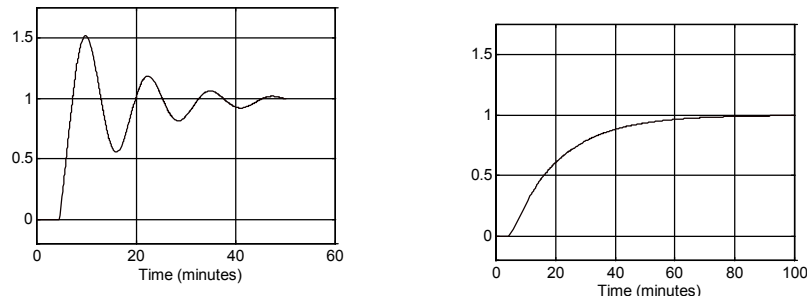


Figure (2) Underdamped (LHS) and overdamped (RHS) system response to a unit change in setpoint (PI control).

Terms used to describe underdamped response characteristics are,

- **Overshoot:** this is the magnitude by which the controlled variable 'swings' past the setpoint. 5/10% overshoot is normally acceptable for most loops.
- **Rise time:** the time it takes for the process output to achieve the new desired value. One-third the dominant process time constant would be typical.
- **Decay ratio:** this is the ratio of the maximum amplitude of successive oscillations.
- **Settling time:** the time it takes for the process output to die to between, say +/- 5% of setpoint.

These characteristics are often used as objectives during a tuning exercise.

Regulatory Control

For a unit step change in the dv, the following type of response profile may be desired,

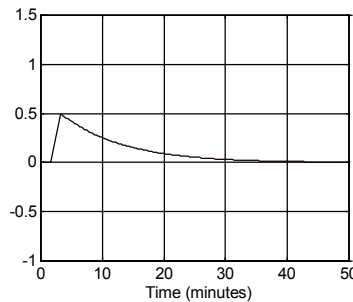


Figure (3) Disturbance rejection (a typical response profile)

i.e. the disturbance initially causes the process to move away from the desired value (which is set to zero in this figure). The controller then adjusts the mv so that the cv slowly moves back to setpoint. In other words the impact that the disturbance has on the closed loop system is eliminated and the system returns to the desired value. A transfer function that could be used to model this behaviour is,

$$\frac{cv(s)}{dv(s)} = \frac{\lambda s}{\lambda s + 1} \quad (6)$$

where the constant λ models the 'peak' effect of the disturbance as well as the speed at which the system returns to steady-state.

Tuning Rules

Rules of thumb

The following rules of thumb are intended to give “ball-park” figure controller settings. The settings⁽¹⁾ assume a series algorithm, the others are for ideal PID

Loop Type	PB(%)	I (mins)	D (mins)
Liquid level	< 100	10	-
Temperature	20 - 60	2 - 15	I/4
Flow	150	0.1	-
Liquid Pressure ⁽¹⁾	50 - 500	0.005 - 0.5	-
Gas Pressure ⁽¹⁾	1- 50	0.1 - 50	0.02 - 0.1
Chromatograph ⁽¹⁾	100 - 2000	10 - 120	0.1 - 20

Often, with level systems exact setpoint following is not essential, hence proportional control is often used. Temperature loop dynamics can be slow because of process heat transfer lags. Deadtime is possible, especially in heat exchangers and temperature is not normally noisy. Consequently PID control is normally preferred. Flow loop dynamics are generally fast (of the order of seconds). Control valve dynamics are normally the slowest in the loop. Flow systems are noisy. However, noise can often be dealt with simply by reducing the gain.

Ziegler Nichols closed loop method

The method is straightforward. First, set the controller to P mode only. Next, set the gain of the controller (k_c) to a small value. Make a small setpoint (or load) change and observe the response of the controlled variable. If k_c is low the response should be sluggish. Increase k_c by a factor of two and make another small change in the setpoint or the load. Keep increasing k_c (by a factor of two) until the response becomes oscillatory. Finally, adjust k_c until a response is obtained that produces continuous oscillations. This is known as the ultimate gain (k_u). Note the period of the oscillations (P_u). The control law settings are then obtained from the following table,

	k_c	T_i	T_D
P	$k_u/2$		
PI	$k_u/2.2$	$P_u/1.2$	
PID	$k_u/1.7$	$P_u/2$	$P_u/8$

Practical use of the technique

It is unwise to force the system into a situation where there are continuous oscillations as this represents the limit at which the feedback system is stable. Generally, it is a good idea to stop at the point where some oscillation has been obtained. It is then possible to approximate the period (P_u) and if the gain at this point is taken as the ultimate gain (k_u), then this will provide a more conservative tuning regime.

Cohen - Coon

This method depends upon the identification of a suitable process model (plant identification has been covered in previous lectures). Cohen-Coon recommended the following settings to give responses having $\frac{1}{4}$ decay ratios, minimum offset and other favourable properties,

	k_c	T_i	T_D
P	$\frac{1}{k_p} \frac{\tau}{\theta} \left(1 + \frac{\theta}{3\tau}\right)$		
PI	$\frac{1}{k_p} \frac{\tau}{\theta} \left(\frac{9}{10} + \frac{\theta}{12\tau}\right)$	$\theta \frac{30 + 3(\theta/\tau)}{9 + 20(\theta/\tau)}$	
PID	$\frac{1}{k_p} \frac{\tau}{\theta} \left(\frac{4}{3} + \frac{\theta}{4\tau}\right)$	$\theta \frac{32 + 6(\theta/\tau)}{13 + 8(\theta/\tau)}$	$\theta \frac{4}{11 + 2(\theta/\tau)}$

In the table k_p is the process gain, τ the process time constant and θ the process time delay.

Practical use of the technique

If the process delay is small (in the limit as it approaches zero) increasingly large controller gains will be predicted. The method is therefore not suitable for systems where there is zero or virtually no time delay.

Direct synthesis

This is a model based tuning technique. It uses an identified process model in conjunction with a user specified closed loop response characteristic. An advantage of this approach is that it provides insight into the role of the 'model' in control system design. A disadvantage of the approach is that a PID controller may not be realised unless an appropriate model form is used to synthesise the control law.

Tuning for servo control

Let the symbol G_p represent the process dynamics and G_c the controller dynamics. If all other dynamic elements within the loop are ignored then the following closed loop transfer function can be derived,

$$\frac{cv}{SP} = \frac{G_c G_p}{1 + G_c G_p} \quad (6)$$

this can be re-arranged to give an expression for the feedback control law as,

$$G_c = \frac{1}{G_p} \left(\frac{\frac{cv}{SP}}{1 - \frac{cv}{SP}} \right) \quad (7)$$

In other words, the controller comprises the inverse of the process model (common to model based design techniques) as well as a specification for the closed loop response characteristic, cv/SP .

A process model can be obtained through plant identification. The closed loop response characteristic, cv/SP must be specified. A simple specification is,

$$\frac{cv}{SP} = \frac{1}{\lambda s + 1} \quad (8)$$

λ is a user specified closed loop time constant.

Substituting this into equation (7) and re-arranging gives,

$$G_c = \frac{1}{G_p} \left(\frac{1}{\lambda s} \right) = \frac{\tau_p s + 1}{k_p \lambda s} = \frac{\tau_p}{k_p \lambda} \left(1 + \frac{1}{\tau_p s} \right) \quad (9)$$

where it has been assumed that the process transfer function is,

$$G_p(s) = \frac{k_p}{\tau_p s + 1} \quad (10)$$

ie. first order, no dead-time.

Based on this process description, the ideal form of a PI controller results, where,

$$k_c = \frac{\tau_p}{k_p \lambda} \text{ and } T_i = \tau_p \quad (11)$$

What do you do if you want derivative action? The first order model results in a control law that is of the PI type. If you wish to synthesis a PID controller, there are two options

- choose $T_D = T_i/4$

- model the process using a 2nd order transfer function.

Revision Exercise

Starting with a second order process transfer function show that a PID control structure can be developed using the direct synthesis derivation technique. What are the settings of the PID controller (in terms of the coefficients of the second order process transfer function) ?

Systems with time delay

Throughout this course, our basic assumption has been that we can model systems using the following transfer function,

$$G_p(s) = \frac{k_p e^{-s\theta}}{\tau_p s + 1} \quad (12)$$

i.e. a first order plus dead-time transfer function. If this were the case, what type of control law would result using the direct synthesis procedure?

Following the derivation presented, the following control law results,

$$G_c = \frac{1}{G_p} \left(\frac{e^{-s\theta}}{\lambda s + 1 - e^{-s\theta}} \right) \quad (13)$$

Note that the following response specification was used (as the time delay cannot be removed from the process),

$$\frac{cv}{SP} = \frac{e^{-s\theta}}{\lambda s + 1} \quad (14)$$

The control law, equation (13) is of non-standard form because of the time-delay terms. Suppose that $e^{-s\theta}$ is approximated by a 1st order Taylor series expansion, i.e.

$$e^{-s\theta} \approx 1 - \theta s$$

Substituting into the denominator of equation (13) and re-arranging gives,

$$G_c = \frac{1}{G_p} \left(\frac{e^{-s\theta}}{(\lambda + \theta)s} \right) \quad (15)$$

It is not necessary to approximate the time delay in the numerator of equation (13) as this is cancelled by an identical term in the process transfer function, $G_p(s)$ giving,

$$G_c = \frac{\tau_p s + 1}{k_p (\lambda + \theta)s} = \frac{\tau_p}{k_p (\lambda + \theta)} \left(1 + \frac{1}{\tau_p s} \right) \quad (16)$$

which is the form of an ideal PI controller where,

$$k_c = \frac{\tau_p}{k_p(\theta + \lambda)} \text{ and } T_i = \tau_p \quad (17)$$

Note the intuitive nature of the controller gain calculation: as the process time delay increases the controller gain will decrease.

Tuning for regulatory control

With reference to the closed loop block diagram, for regulatory control the following closed loop transfer function may be derived,

$$\frac{cv}{dv} = \frac{1}{1 + G_c G_p} \quad (18)$$

This closed loop expression can be re-arranged to give an expression for the feedback control law as,

$$G_c = \frac{1}{G_p} \left(\frac{1 - \frac{Y}{d}}{\frac{Y}{d}} \right) \quad (19)$$

Again, the controller consists of the inverse of the process model as well as a specification for the closed loop response characteristic, cv/dv .

The process model is obtained through plant identification however, the closed loop response characteristic, cv/dv , must be specified by the designer. Using the simple specification described earlier,

$$\frac{cv(s)}{dv(s)} = \frac{\lambda s}{\lambda s + 1} \quad (20)$$

where λ is user specified. Substituting this into equation (19) and re-arranging gives,

$$G_c = \frac{1}{G_p} \left(\frac{1}{\lambda s} \right) \quad (21)$$

This is exactly the same form as equation (9) for servo control. Hence the controller gain and integral term for a PI controller is given by,

$$k_c = \frac{\tau_p}{k_p \lambda} \text{ and } T_i = \tau_p \quad (22)$$

Final Remarks

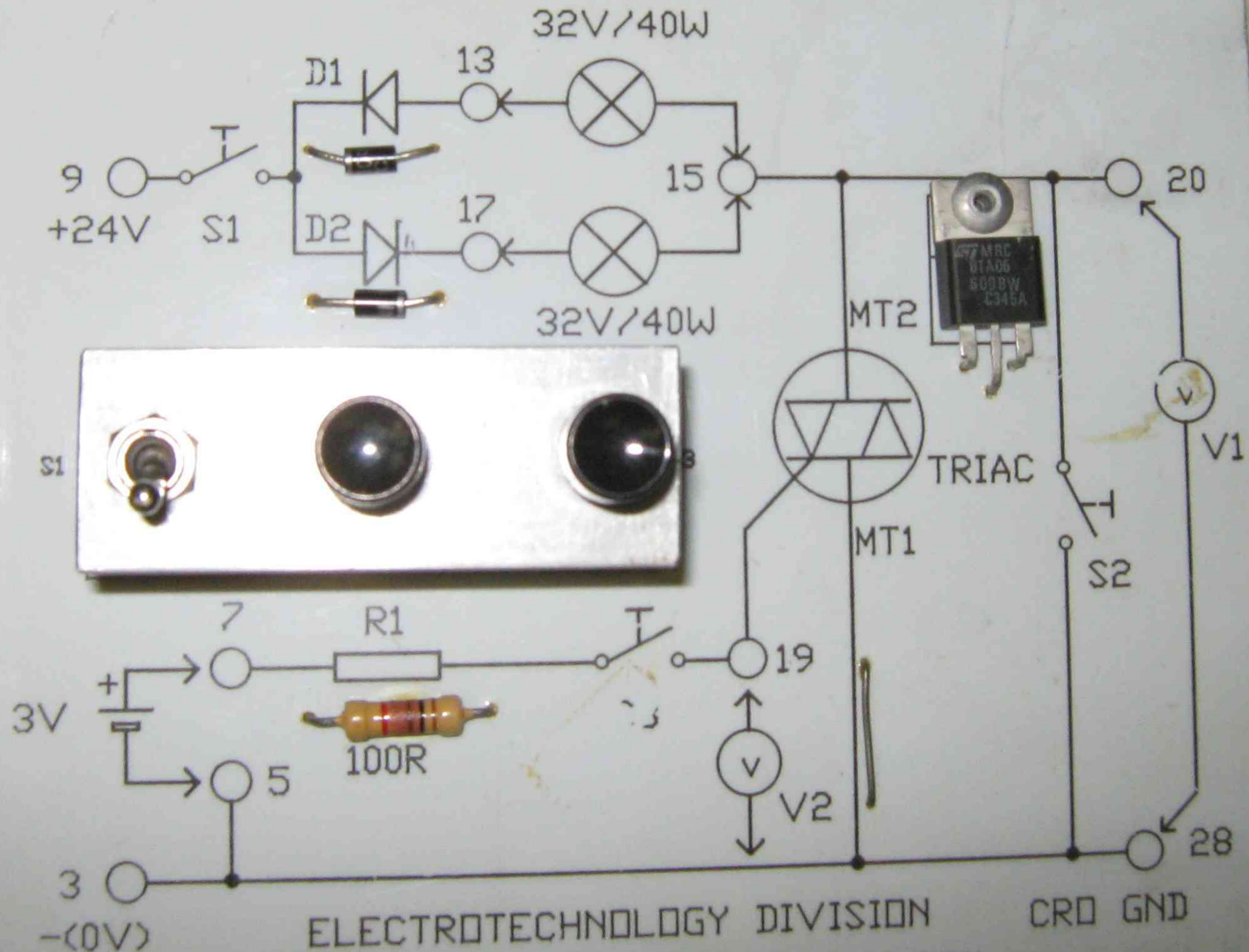
The notes have reviewed PID control, discussed the modes of the various control algorithms, the different structures of algorithms that exist and standard tuning rules. The tuning rules reviewed include, Ziegler-Nichols, Cohen-Coon, and direct synthesis. Remember:

- the tuning rules are only valid for the 'ideal' PID control structure and any prediction of control law settings should be adjusted if an alternative PID implementation is used.
- the tuning rules are only valid for self-regulating processes (i.e open loop stable processes such as those that may be described by the 1st order plus dead-time description).

Luckily most process systems are self-regulating the exception to the rule being level systems. Tuning of level controllers will be the subject of the next section of the notes.

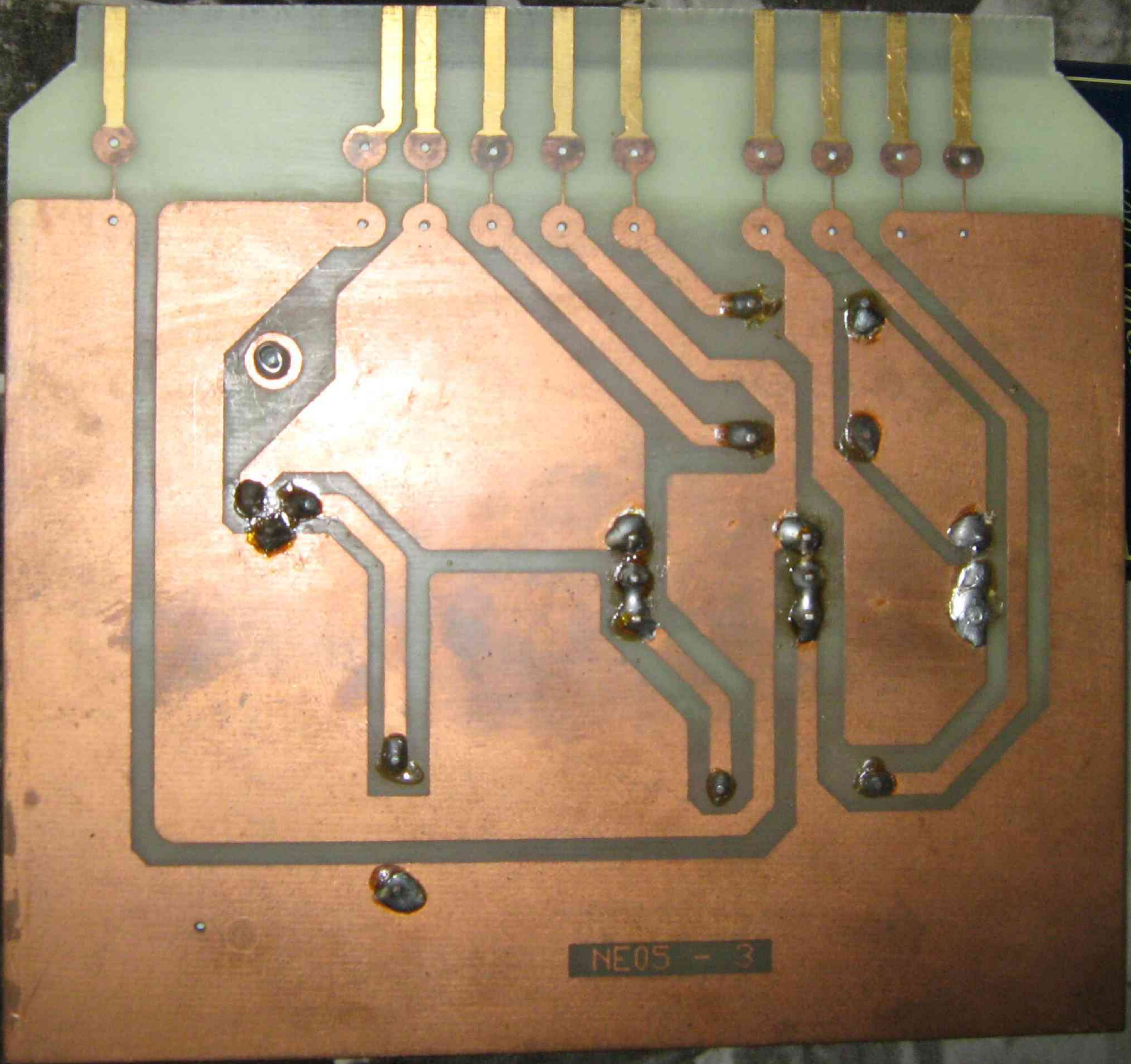
POWER CONTROL DEVICES

NE05 SECTION 3



ELECTROTECHNOLOGY DIVISION
SYDNEY INSTITUTE OF TECHNOLOGY

CR0 GND



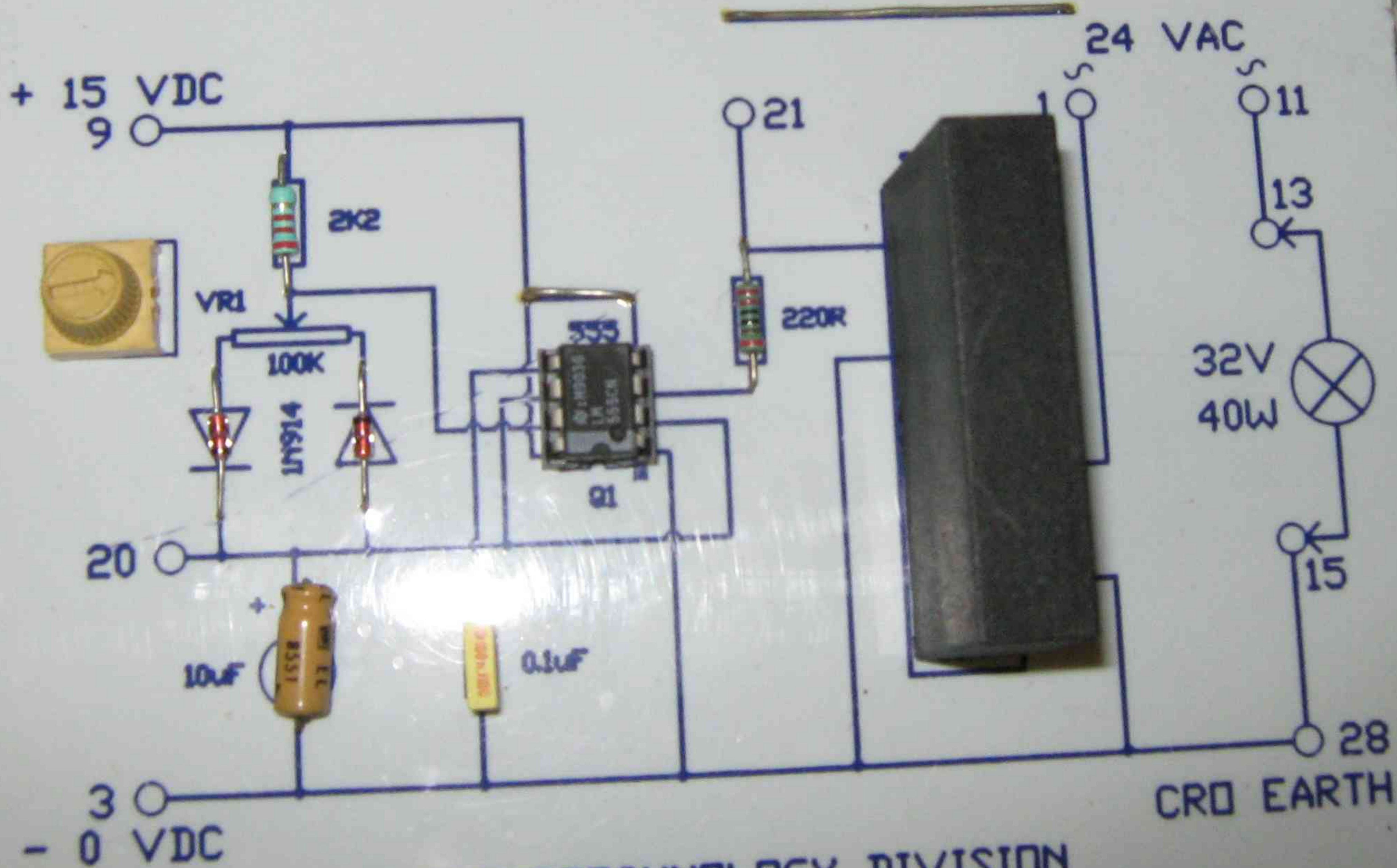
Library of Congress
1907 Calendar

WAR

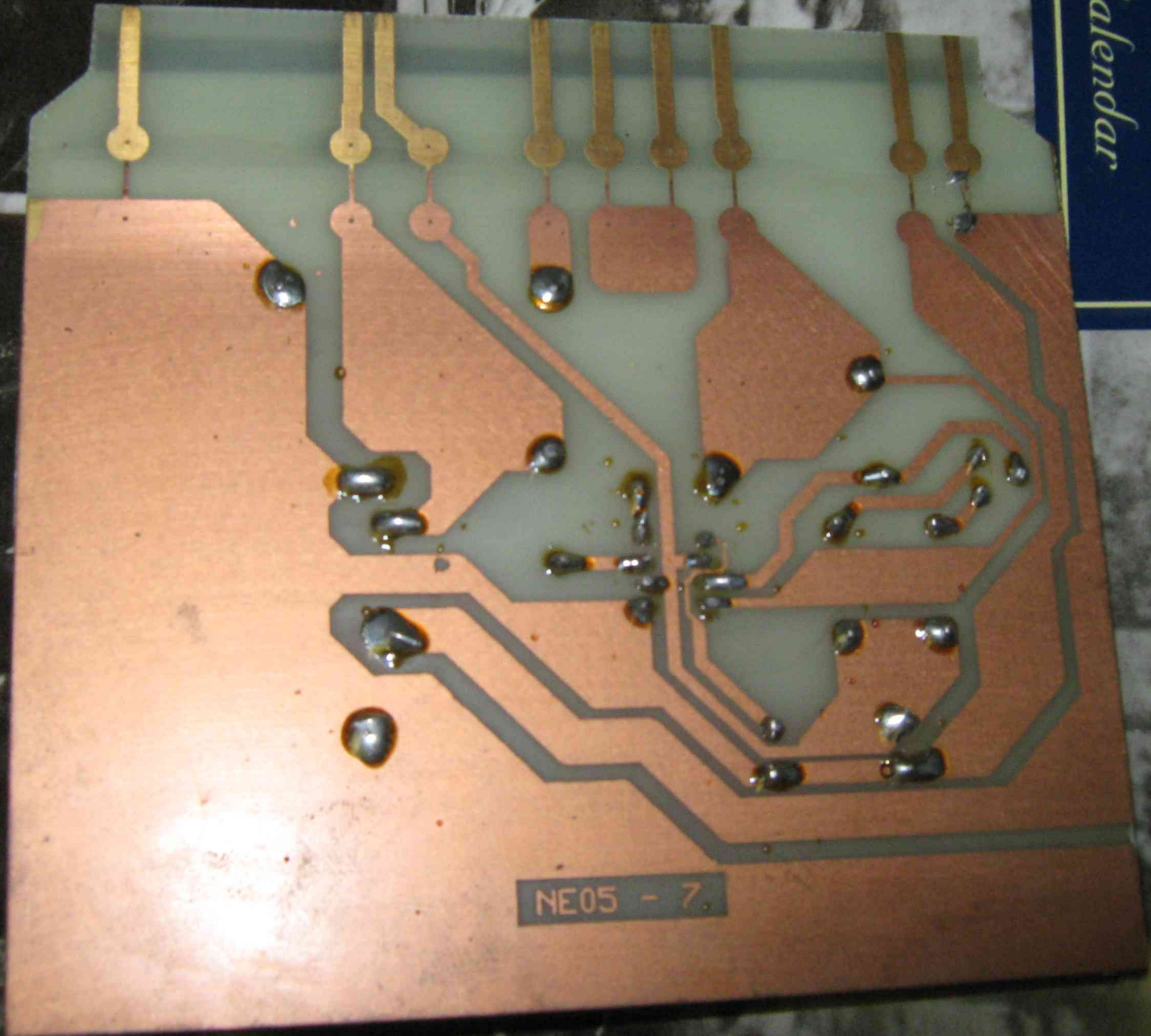
POWER CONTROL DEVICES

NEOS SECTION 7

ZERO VOLTAGE SWITCHING



ELECTROTECHNOLOGY DIVISION
SYDNEY INSTITUTE OF TECHNOLOGY

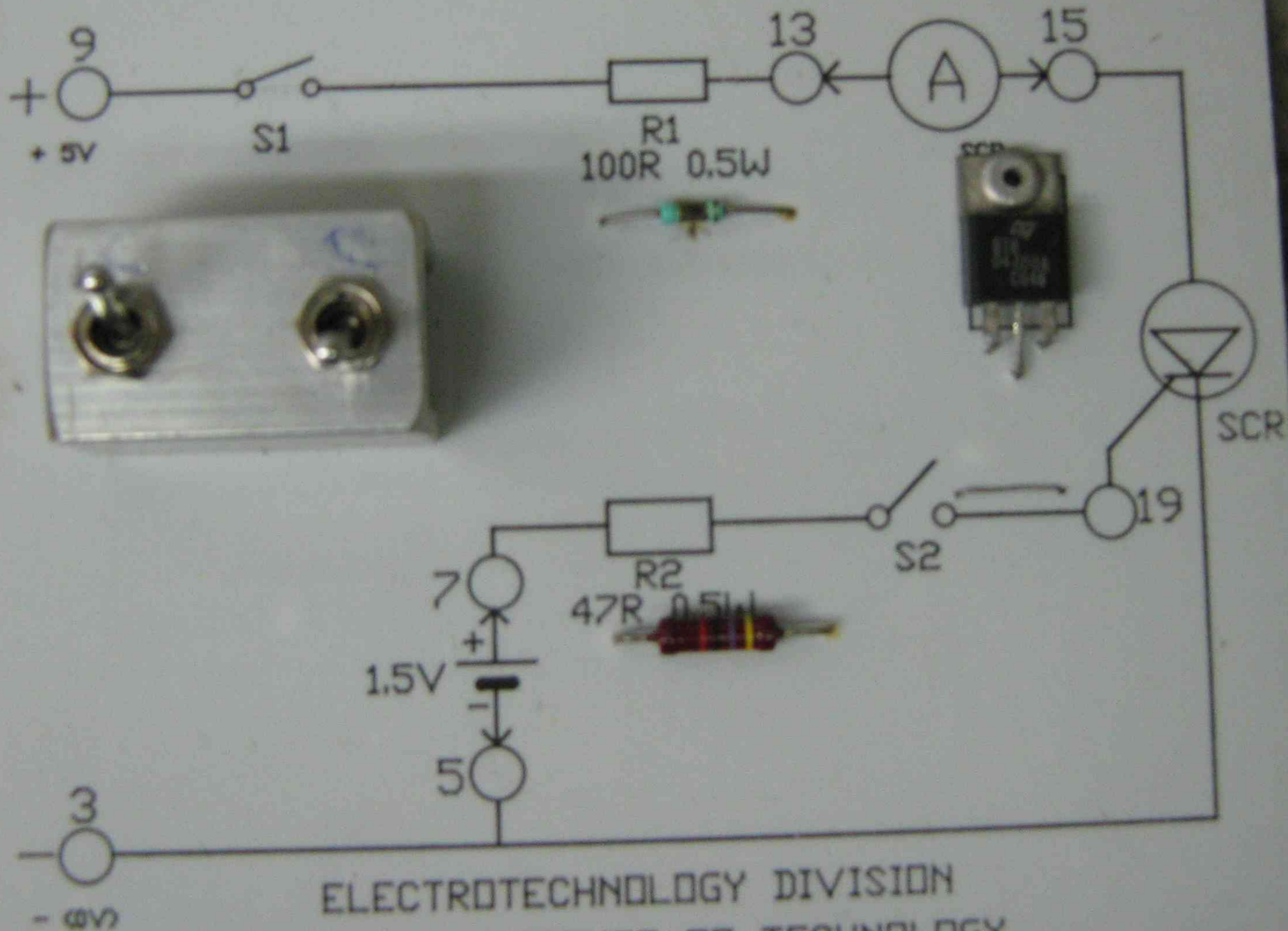


NE05 - 7.

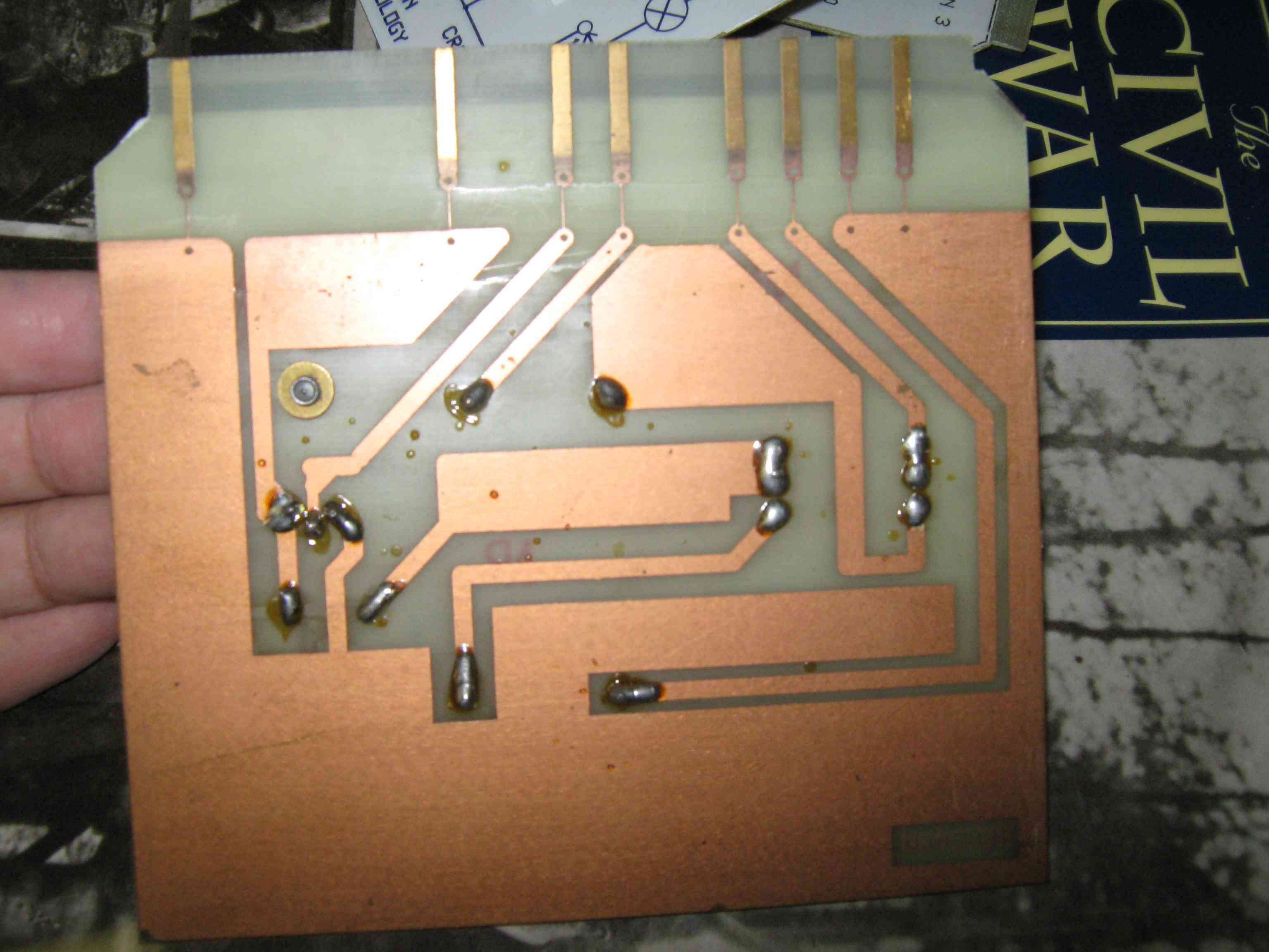
YEAR
ary of Congress
07 Calendar

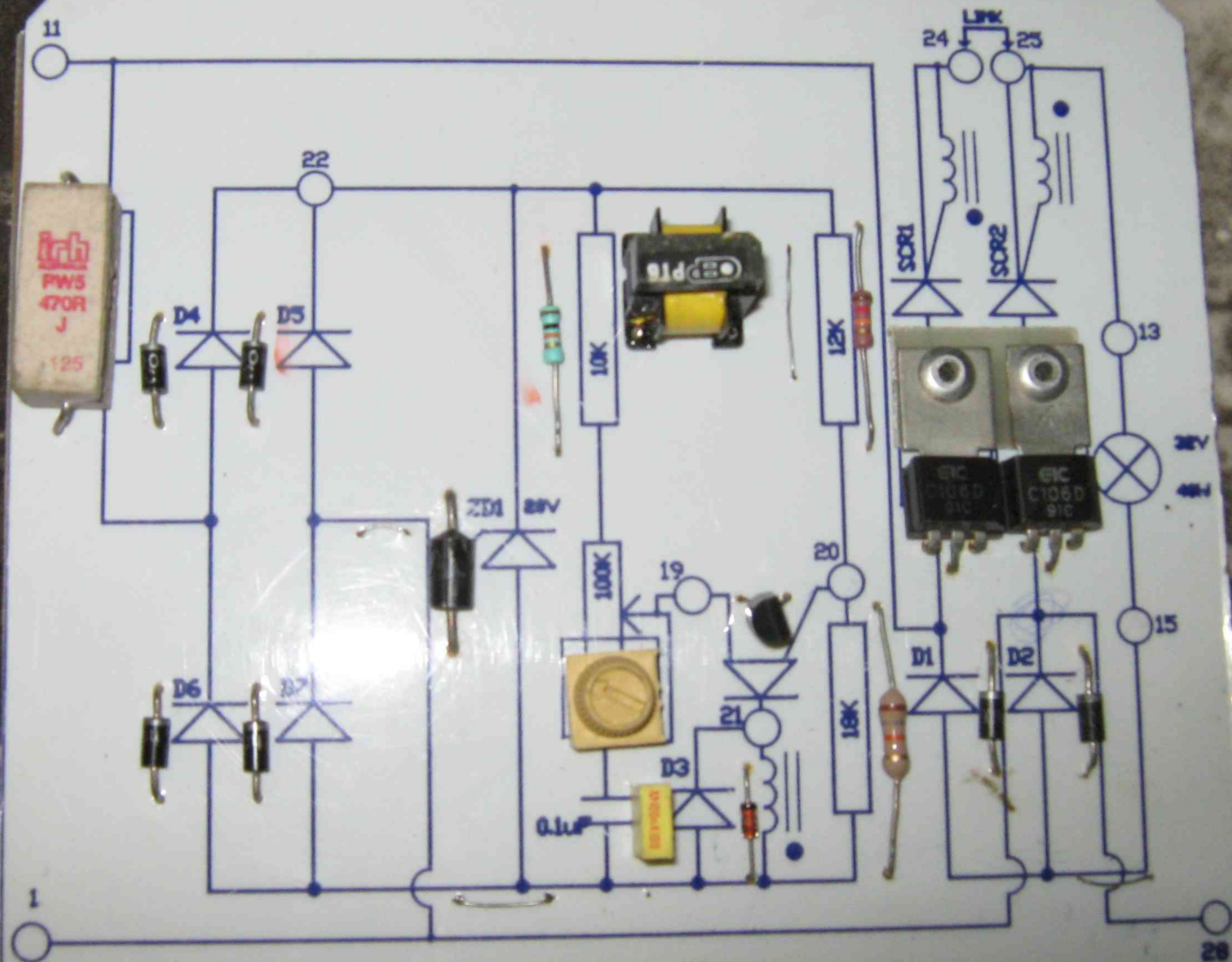
POWER CONTROL DEVICES

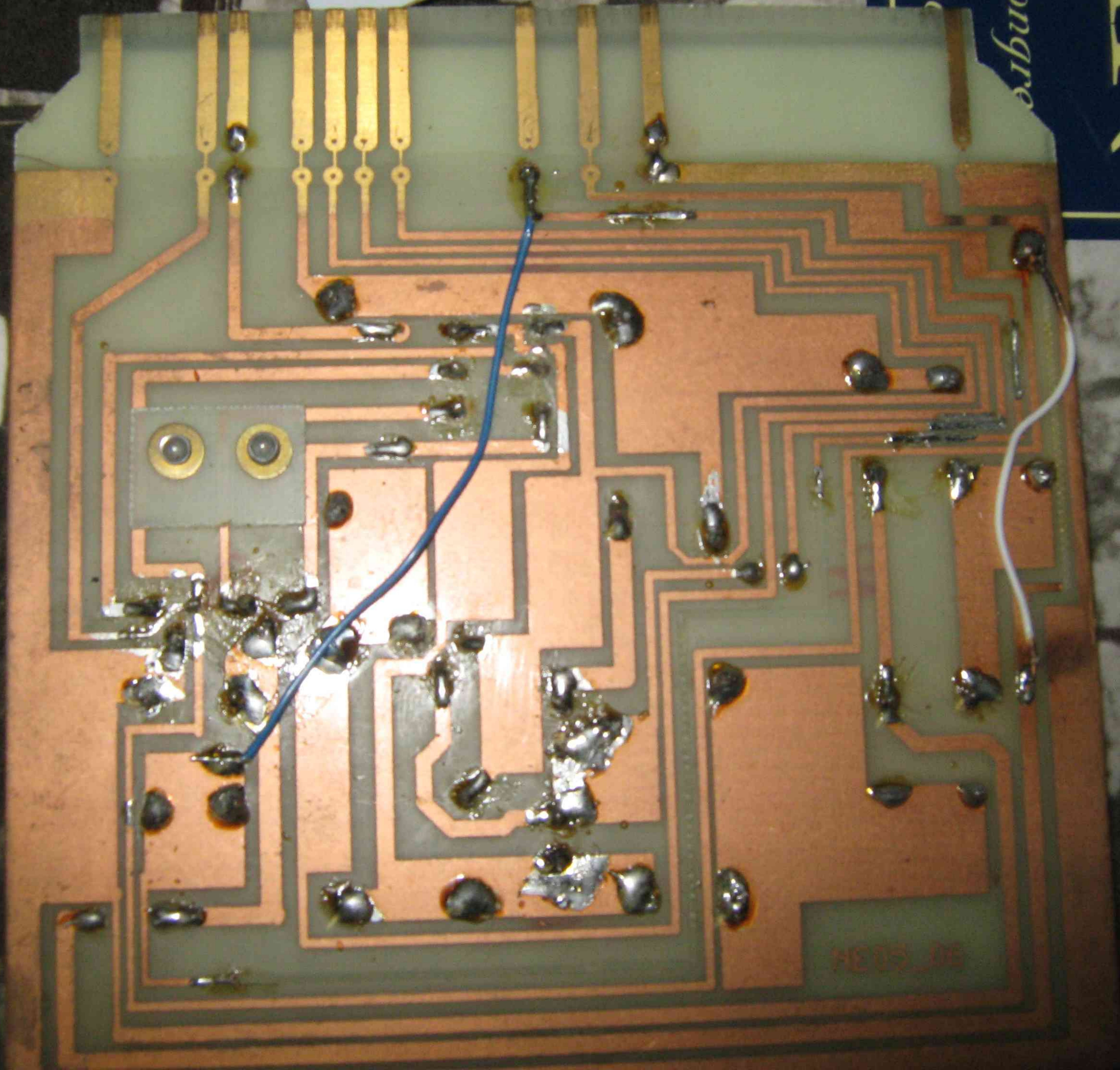
NE05 SECTION 2B



ELECTROTECHNOLOGY DIVISION
SYDNEY INSTITUTE OF TECHNOLOGY

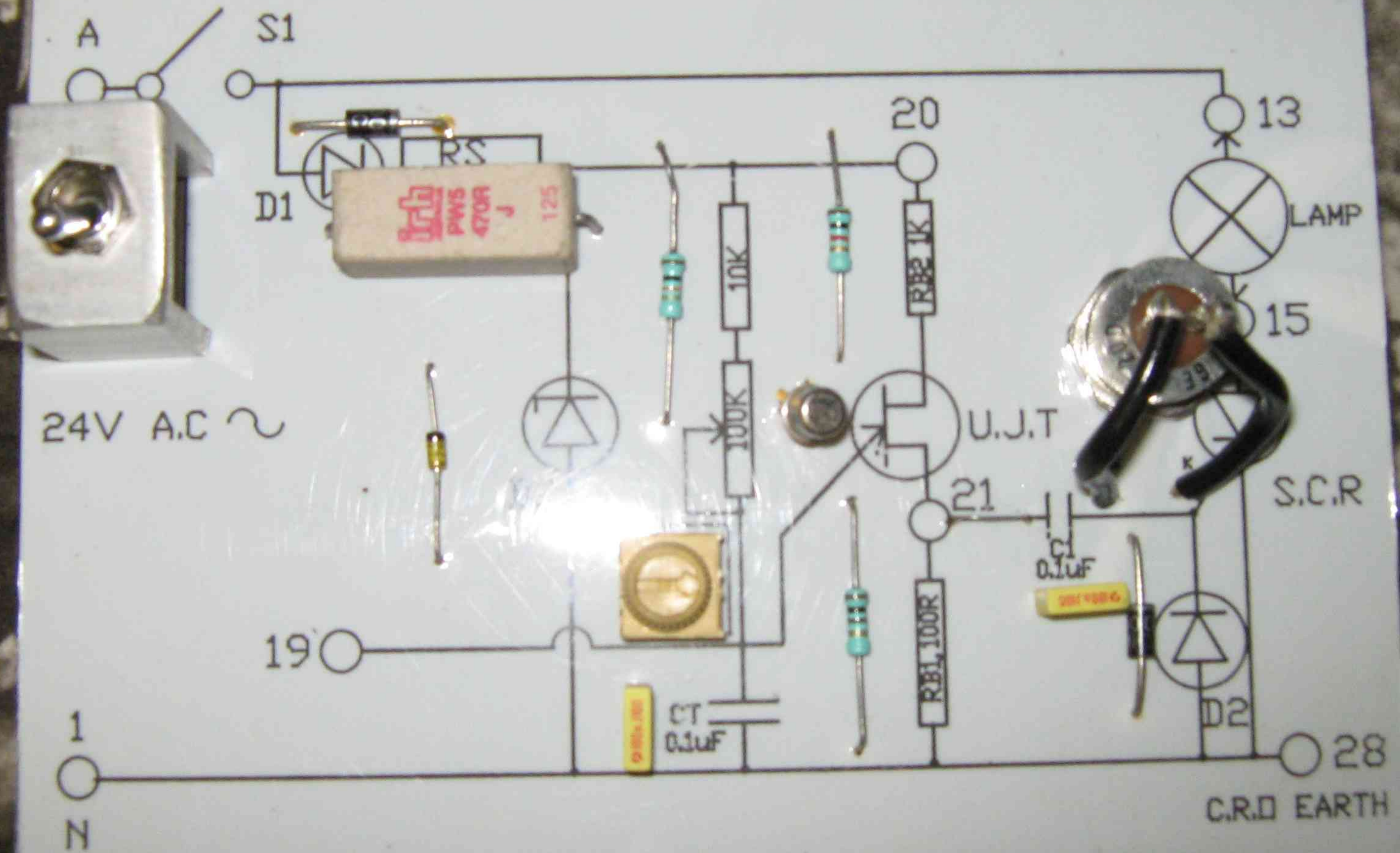






POWER CONTROL DEVICES NE05_4B

U.J.T TRIGGER PULSE
GENERATOR - MAINS
SYNCHRONISATION

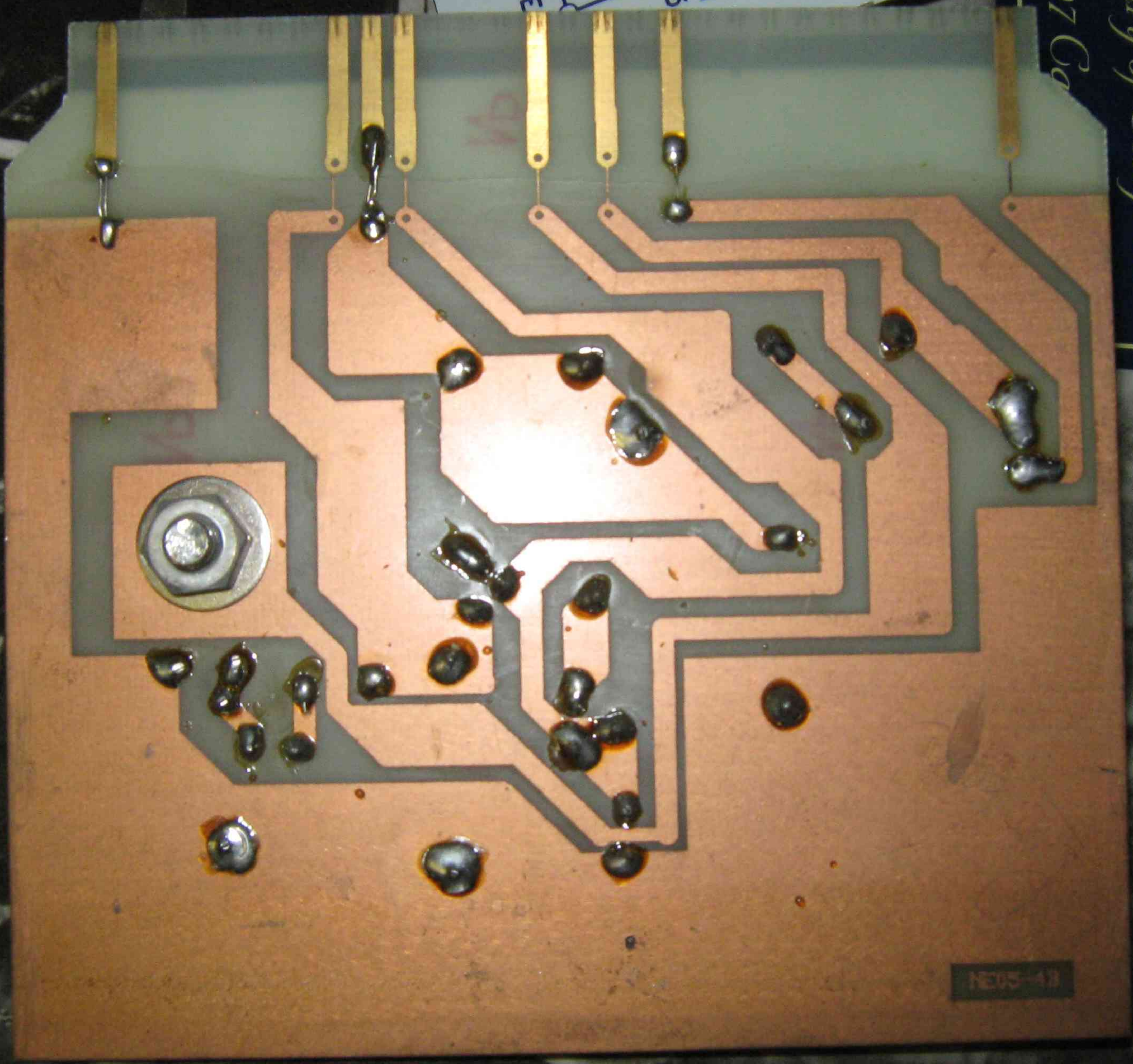


ELECTROTECHNOLOGY DIVISION
SYDNEY INSTITUTE OF TECHNOLOGY

WARR

ary of Congress

997 Ca

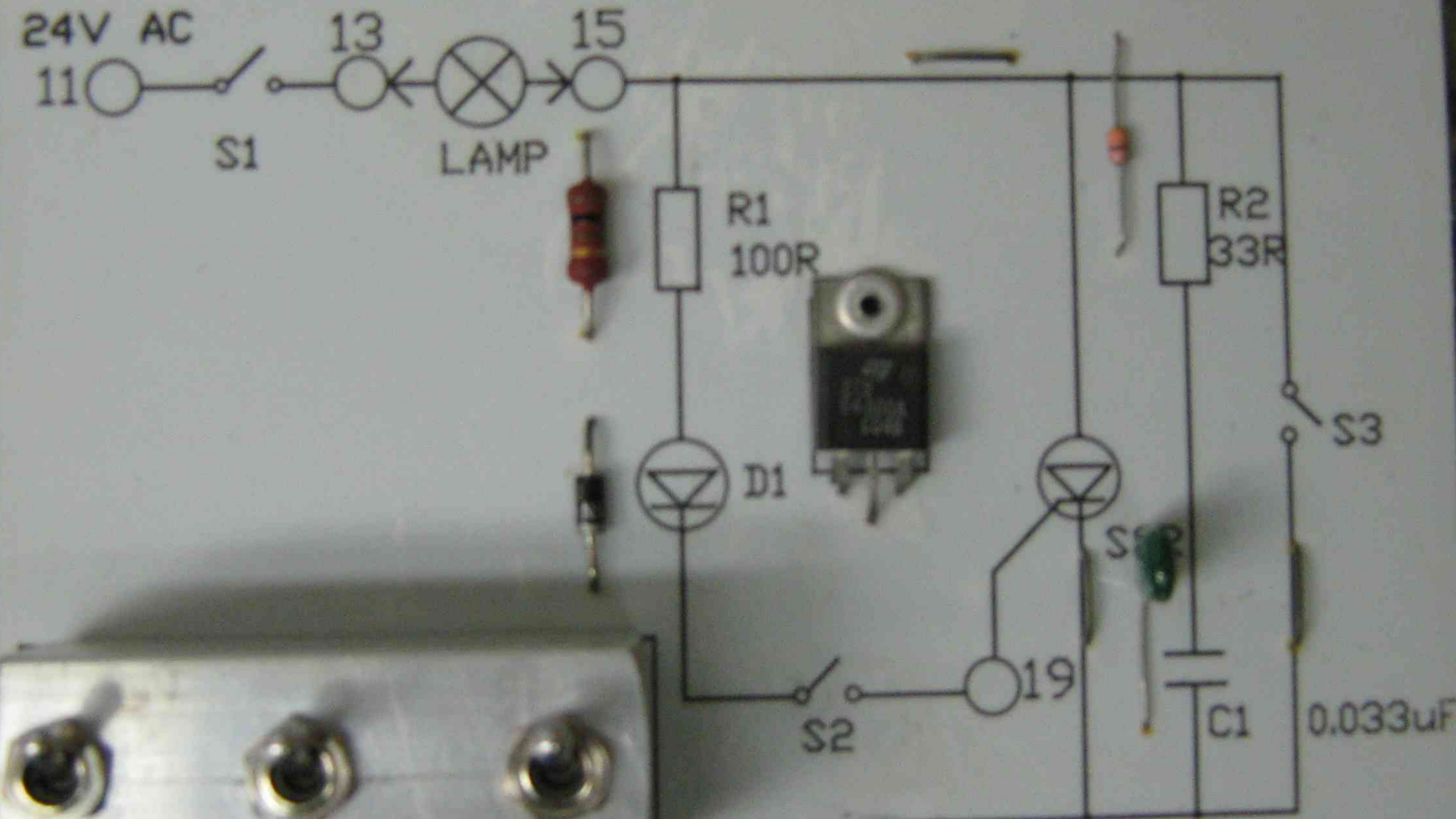


NEUS-41



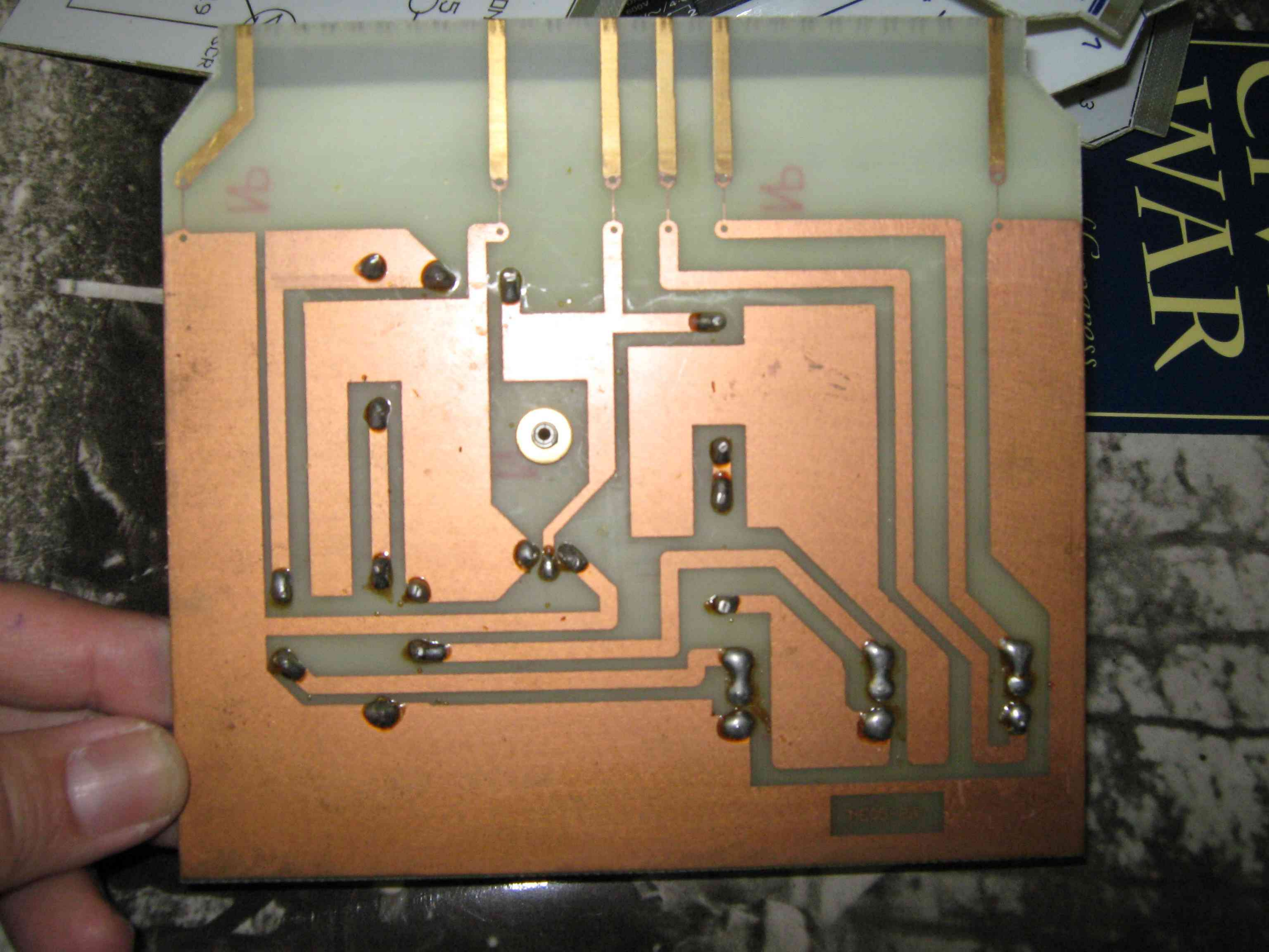
POWER CONTROL DEVICES

NE05 SECTION 2A



ELECTROTECHNOLOGY DIVISION

SYDNEY INSTITUTE OF TECHNOLOGY



9
SCR

16

16

WEAR

CP mass