MICROPROCESSORS

A summary by
Hamza Younes

Editor's Note:

--Midterm:

Short Questions (Flag status questions, Memory Status, Exponent & Mantissa, String Instructions) are not included in this summary

Midterm is usually split into two parts (as shown in the attached Midterm Exam):

Part 1: Short questions (50%)
Part 2: Code (50%)

Study well for the midterm as it is the easiest part to gain marks in out of the 100% of the course. The material may look difficult at first but with hard work you'll eventually get a hold of it. This in no way means that the course is difficult nor is it easy but with hard work and perseverance nothing is impossible. May God grant you Success in both the exam and life 💔

Flag status questions, Memory Status, Exponent & Mantissa, الاسئلة القصيرة (*String Instructions* غير موجودة بهذا التلخيص)

الميدتيرم كما هو مبين بالنموذج المرفق مقسم الى جزئين الاول اسئلة قصيرة و الثاني اسئلة كود

عزيزي القارئ ادرس جيدا للميدتيرم لانه احسن فرصة تجمع فيه علامات من 100 .المادة قد تبدو غريبة بالاول لكن مع الوقت و الجهد بتصير في متناول اليد لا يعني ذلك انها صعبة ولا سهلة و لكن لكل مجتهد نصيب ربنا لا يضيعلكم تعب و يوفقكم في امتحاناتكم و دنياكم ❤️

--Chapter 9 (Quiz):

Make sure to know each pin
output/input
which mode it operates in (min or max)
What the function of each pin is

Make sure to study the Bus Timing Question well as it makes up 50% of the quiz. The other 50% is MCQ (Pins, their functionality and type) and a question to draw either clock or 8088/86 buffering & demultiplexing

make sure to study well for the quiz as its grade determines if you pass or fail.

اعرف كل pin نوعها والmode الي بتتبع اله و وظيفتها. اهم ما في هذا التشابتر هو سوال ال bus timing عليه نصف علامة الكويز باقي الكويز بكون رسم رسمة من الرسمات المطلوبة ودوائر عن ال pins و وظائفهم. عزيزي القارئ لا تستهن بالكويز لأنه نجاحك في المادة يعتمد عليه

--Final (Ch10 & Ch11):

Shadowing (Partial Decoding) is not included in this Summary

Memorize The Codes and Drawings of Ch11 50% of the final is on them, 25% is MCQ on the Midterm's material and the remaining 25% is on Ch10

ال(Shadowing (Partial Decoding غير موجود بهذا التلخيص

احفظ اكواد ورسمات تشابتر 11 الموجودين في التلخيص لأنه نصف الفاينل عليهم باقي الفاينل ربع دوائر على مادة الميد والربع الاخر سؤال على تشابتر 10

ARCHITECTURE OF 8086

GENERAL PURPOSE REGISTERS

Some registers can store 8 bits and some can store 16 bits.

8-Bit	AH	AL	BH	BL	CH	CL	DH	DL
16-Bit	AX	BX	CX	DX				

FLAG REGISTERS

X	X	X	X	OF	DF	IF	TF	SF	ZF	X	AC	X	PF	X	CF

→ OF (Overflow flag) : used in operations that involve signed numbers.

→ SF (Sign flag) : provides sign of result value.

→ ZF (Zero flag) : indicates result is zero

→ PF (Parity flag) : the count of 1's in a number (0 for odd, 1 for even)

→ AC (Auxiliary Carry) : holds the half carry (temporary carry)

SEGMENT REGISTERS OF MEMORY POINTERS

OPERATION	SEGMENT REGISTER	OFFSET REGISTER (memory pointer)
Reading instruction code from memory	CS	IP
Physical address of stack top memory address	SS	SP
Data transfer between memory & microprocessor	SS (CS, DS, ES)	BP
Data transfer between microprocessor & memory	DS (CS, ES, SS)	BX , SI , DI
Data transfer between microprocessor & memory using direct addressing mode	DS (CS, ES, SS)	EA
Data transfer using string instruction SOURCE ⟶ DESTINATION ⟶	DS ← مصدر ES ← الى	SI ← مصدر DI ← الى

LOGICAL ADDRESS ⟶ REGISTER (CS, SS, DS, ES) : OFFSET (IP, SP, BP, BX, SI, DI, EA)

OFFSET ADDRESS ⟶ OFFSET

PHYSICAL ADDRESS ⟶ REGISTER + OFFSET

ADDRESSING MODES

THE "Mov" INSTRUCTION

MOV DESTINATION, SOURCE

NEVER USE CS AS A DESTINATION

copies data from source to destination, data in source doesn't change
contents of destination change for all except CMP & TEST instructions.

→ MOV is for byte or word sized data ←

BOTH OPERANDS MUST BE THE SAME SIZE (16 to 16) (8 to 8)

no memory to memory / no immediate segment moves / No segment to segment (ES→DS)

Summary of MOV

MOV REG, memory	MOV REG, REG	MOV REG, immediate
MOV memory, REG	MOV memory, immediate	MOV To, from

REG: AH, AL, BH, BL, CH, CL, DH, DL, AX, BX, CX, DX
memory: [BX], [BX+SI+7], variable, etc...
immediate: 5, -24, 3Fh, 10001101b, etc...

IN, OUT INSTRUCTION

IN transfers data from an external port to AL or AX — IN AL, port
OUT transfers data from AL or AX to external port — OUT port, AL

VARIABLES

`name DB value`	Define byte (8-bit)
`name DW value`	Define word (16-bit)

→ Decimal data is represented as is (MOV AL, 100)
→ Hex data ends with an H (MOV AL, 2H) and if it starts with a letter a 0 is put before the letter (MOV AL, 0F2H), binary data ends with ab (MOV AL, 1011b).

name: should start with letter value: can be any hex, binary or decimal value or "?" for uninitialized variables

STACK → ONLY ON 16 BIT VALUES

PUSH - stores 16 bit value in stack

PUSH REG	PUSH memory
PUSH SREG	PUSH immediate

REG: AX, BX, CX, DX

SREG: DS, ES, SS, CS.

memory: [BX], [BX+7], 16 bit variable etc...

immediate: 5, -24, 3Fh, 10001101b, etc...

POP - gets 16 bit value from stack

POP REG	POP SREG
POP memory	

REG: AX, BX, CX, DX

SREG: DS, ES, SS

memory: [BX], [BX+7], 16 bit variable etc...

Stack can be used to restore the original value of a register:

```
MOV  AX, 1234h
PUSH AX    ; stores the value of AX in a stack

MOV AX, 5678h ; modify the value of AX

POP AX ; restores the original value of AX
```

LEA stores the address of a variable in a register

ARRAY

Arrays are a chain of variables. A text string is an example of a Byte array each character is presented as an ASCII code value (0...255)

```
a DB 48h, 65h, 6Ch, 6Ch, 6Fh, 00h
b DB "Hello", 0
```

you can access any element of the array using square brackets

```
MOV AL, a[3]
```

you can use BX as an index if it has a number stored in it

if you need to declare a large array of size n use DUP

```
name DW n DUP (initial value)
```

ARITHMETIC & LOGIC INSTRUCTIONS

ADD, SUB, CMP, AND, TEST, OR, XOR

After operation between operands, result is always stored in first operand
CMP & TEST instructions affect flags only and don't store a result
and are used to make decisions during program execution the following flags are affected: CF, ZF, SF, OF, PF, AF

INSTRUCTION REG, memory (ADD/SUB/CMP/AND/TEST/OR/XOR)	INSTRUCTION memory, REG
INSTRUCTION REG, REG	INSTRUCTION memory, immediate
INSTRUCTION REG, immediate	REG: AX, BX, CX, DX, AH, AL, BH, BL, CH, CL, DH, DL memory: [BX], [BX+7], variable, etc... immediate: 5, -24, 3Fh, 10001101b, etc...

ADD: ADD:

- **ADD** adds second operand to first (A = A + B) — ADD A,B (Carry affects CF)
- **SUB** subtracts second operand from first (A = A - B) — SUB A,B (Borrow affects CF)
- **CMP** subtracts second operand from first for flags only (if (A - B) then) — CMP A,B
- **AND** logical and between all bits of two operands (A = A . B) — AND A,B
- **TEST** the same as AND but for flags only (if (A.B) then) — TEST A,B
- **OR** logical or between all bits of two operands (A = A or B) — OR A,B
- **XOR** logical xor between all bits of two operands (A = A ⊕ B) — XOR A,B

MUL, IMUL, DIV, IDIV

INSTRUCTION REG (MUL/IMUL/DIV/IDIV)	INSTRUCTION memory
REG: AX, BX, CX, DX, AH, AL, BH, BL, CH, CL, DH, DL	memory: [BX], [BX+7], variable, etc...

MUL & IMUL instructions affect **CF & OF** when result is over operand size
These flags are set to 1, when result fits in operand size these flags are set to 0
for DIV & IDIV flags are undefined

MUL unsigned multiply

MUL operand

OPERAND	8-bit (byte)	16-bit (word)
REGISTER	AX = AL * operand	(DX AX) = AX * operand

IMUL signed multiply

IMUL operand

OPERAND	8-bit (byte)	16-bit (word)
REGISTER	AX = AL * operand	(DX AX) = AX * operand

DIV unsigned divide

DIV operand

IDIV signed divide

IDIV operand

OPERAND	8-bit (byte)	16-bit (word)
REGISTER	AL = AX / operand AH = remainder	AX = (DX AX) / operand DX = remainder

INC, DEC, NOT, NEG

INSTRUCTION REG	INSTRUCTION memory
REG: AX, BX, CX, DX, AH, AL, BH, BL, CH, CL, DH, DL	memory: [BX], [BX+7], variable, etc ...

NOT effects no flags reverses all the bits of the operand

INC & DEC effect the following flags ZF, SF, OF, PF, AF

NEG effects the following flags CF, ZF, SF, OF, PF, AF two's complement of operand

---

SHIFT (SHL, SAL, SHR, SAR) bits that are shifted are stored in CF

SHL: moves a 0 into the rightmost bit of a register/memory.

SAL moves a 0 into the rightmost bit of a register/memory.

SHR moves a 0 into the leftmost bit of a register/memory.

SAR copies the sign-bit through the number.

INSTRUCTION (SHL, SAL, SHR, SAR) memory, immediate

INSTRUCTION REG, immediate

operand 1 is shifted by operand 2

Shifting left = x2
Shifting right = ÷2

PROGRAM FLOW CONTROL

READ / WRITE

the interrupt instruction INT 21H is used to determine what action is to be executed based off the value of AH

Value of DL	Value of AH	Action of INT 21H	Stored / found
-	01 H	input char from Keyboard	AL
-	02 H	output char on screen	DL
FFH	06 H	input char from Keyboard without echo	AL
-	06 H	output char on screen	DL
-	09 H	output a $-terminated string	DS:DX

UNCONDITIONAL JUMP

the basic instruction that transfers control to another point in the program.

JMP label below is an example of how it can be used:

```
org 100h
mov ax, 5      ;set ax to 5
mov bx, 2      ;set bx to 2
jmp calc       ; go to calc
back: jmp stop ; go to stop

calc:
add ax, bx     ;add bx to ax
jmp back       ; go back

stop:
ret
```

CONDITIONAL JUMPS

INSTRUCTION	DESCRIPTION	CONDITION	OPPOSITE
JZ, JE	jump if zero, equal	ZF = 1	JNZ, JNE
JC, JB JNAE	jump if carry, below not above equal	CF = 1	JNC, JNB, JAE
JS	jump if sign	SF = 1	JNS
JO	jump if Overflow	OF = 1	JNO
JPE JP	jump if parity even	PF = 1	JZ, JE

the example below is an assembly program that keeps reading keys and displays them until an '@' is entered:-

```
.MODEL small
.CODE
MAIN:
        mov AH,6          ;read a Key without waiting
        mov DL,0FFH
        INT 21H

        JE MAIN           ; if no Key typed go back to
        cmp AL,'@'
        JE MAIN1          ; if an "@" is typed go to

        mov AH,06H
        mov DL,AL
        INT 21H
        jmp MAIN          ; repeat/loop
MAIN1:
        .EXIT             ;exit program
```

LOOPS

INSTRUCTION	JUMP CONDITION & OPERATION	OPPOSITE
LOOP	decrease cx, jump to label if cx not zero (loops until cx is zero)	—
LOOPE	decrease cx, jump to label if cx not zero and equal (ZF = 1)	LOOPNE
LOOPNE	decrease cx, jump to label if cx not zero and not equal (ZF = 0)	LOOPE
LOOPNZ	decrease cx, jump to label if cx not zero and ZF = 0	LOOPZ
LOOPZ	decrease cx, jump to label if cx not zero and ZF = 1	LOOPNZ

loops are basically the same as jumps (conditional jumps + compare)

the following example uses a loop to add two variables X, Y 100 times

```
.MODEL SMALL

.DATA

    X DW 10
    Y DW 5

.CODE

    MOV AX,[X]
    MOV DX,[Y]

    MOV CX,100

L1: ADD AX,DX
    LOOP L1

END
```

DOT (.) COMMANDS

COMMAND	ACTION / DESCRIPTION
.BREAK	terminates a .WHILE or .REPEAT block
.CONTINUE	jumps to the top of a .WHILE or .REPEAT block
.ELSE	begins block of statement to execute when the condition of .IF is false.
.ELSEIF condition	tests condition and executes until .ENDIF or another .ELSEIF is found.
.ENDIF	ends the block of code of .IF, .ELSE, .ELSEIF
.ENDW	ends the block of code of .WHILE
.IF condition	if condition is met (true) block of code will be executed
.REPEAT	repeats block of code until condition in .UNTIL is met
.UNTIL condition	sets condition for .REPEAT (repeats the block of code until condition is true)
.UNTILCXZ	repeats block of code until CX = 0
.WHILE condition	loops block of code between it & .ENDW until condition is false

```
.REPEAT
    statements
.UNTIL condition
```

```
.WHILE condition
    statements
.ENDW
```

```
.IF condition
    statements
.ELSE
    statements
.ENDIF
```

PROCEDURES

procedures are functions in assembly. A procedure is a part of code that can be called from your program in order to perform specific tasks.

```
name PROC
     code statements
     RET
name ENDP
```

ex to use in program →

```
     CALL m1
     MOV AX, 2

m1 PROC
     MOV BX, 5
     RET
m1 ENDP
```

you can offset a procedure into a register and call the register instead of the procedure the following program displays "OK" using a procedure called "DISP" that is stored in BX.

```
.MODEL SMALL
.CODE
     MOV BX, OFFSET DISP
     MOV DL, '0'
     CALL BX
     MOV DL, 'K'
     CALL BX
.EXIT

DISP PROC NEAR
     MOV AH, 2
     INT 21H
     RET
DISP ENDP
END
```

MACROS

macros are procedures with parameteres and are usually declared above the code it's being used by

```
name MACRO parameters
     code statements
ENDM
```

MIDTERM QUICK REVISION

HOW ARE VARIABLES STORED

Step ① convert number to binary

Step ② place a decimal point next to the first "1" from the left
number will become 1.rest of it $\times 2^{\#\text{digits right of decimal}}$

Step ③ find exponent 127 + (# of digits right of decimal)

Step ④

Sign	exponent	FRACTION

SUMMARY OF REGISTER USAGES

AH	AL	BH	BL	CH	CL	DH	DL
1) stores remainder 2) Determines INT 21H	1) Stores from port 2) Stores value to be multiplied 3) Stores quotient 4) Stores from Keyboard when AH 01H						1) Value is output on screen when AH 02H

AX	BX	CX	DX
1) stores value to be multiplied 2) Stores quotient		1) is decremented in loop	1) Stores remainder

(CH9)

HARDWARE SPECS 88/86 (8 bit) (16 bit)

OPERATION MODES:

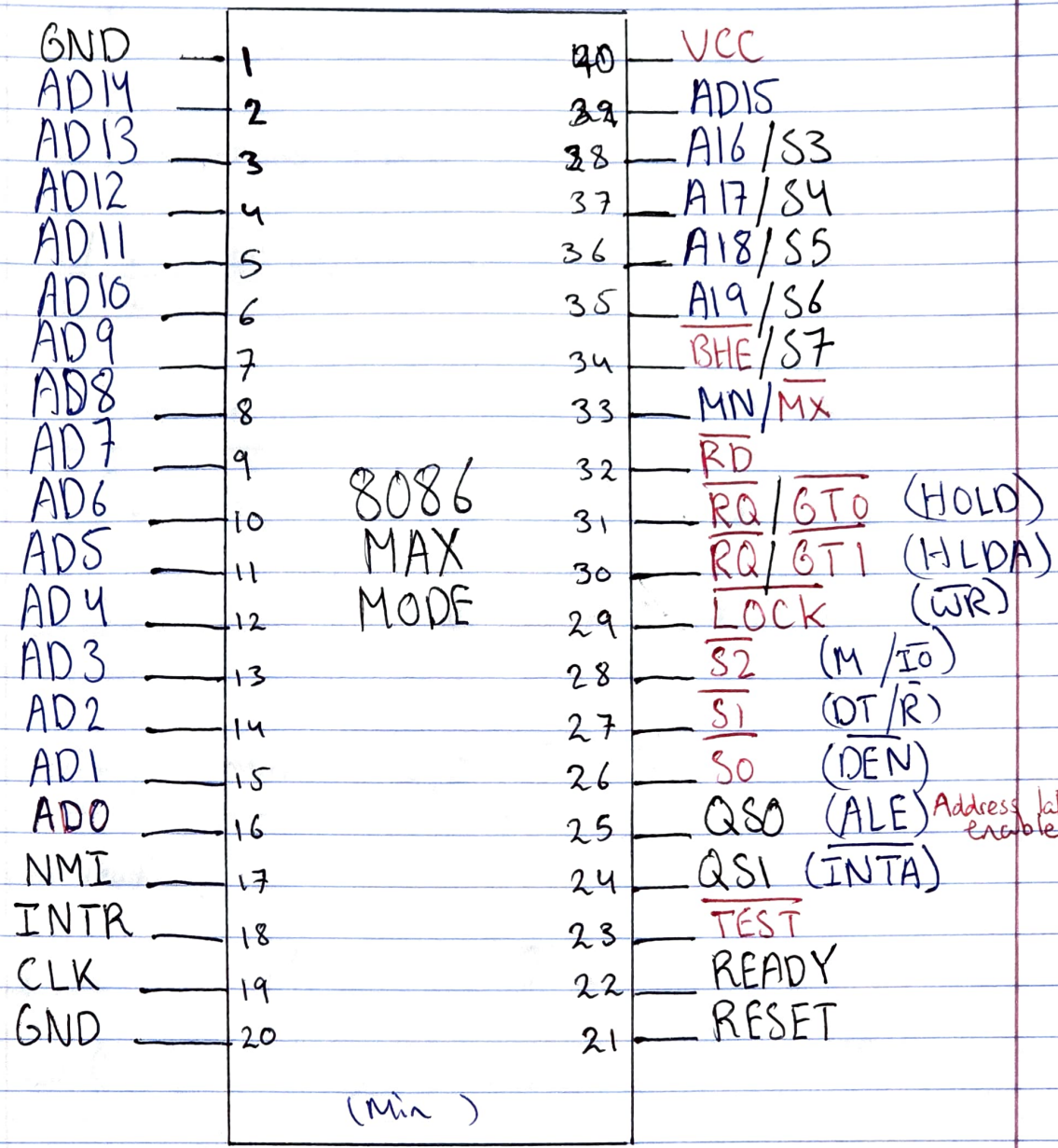
→ Minimum Mode: the microprocessor generates the necessary control signals

→ Maximum Mode: an additional bus controller is required to generate the control signals

a good example would be the difference between cargo planes and passenger planes. cargo planes (Minimum) don't need ~~stairs~~/food/catering while passenger planes (Maximum) need everything (~~stairs~~/food/catering/bathrooms)

PIN LAYOUT 8086

numbering ↑



1,20 : GND

16-2 : AD0-AD14

17: NMI

18: INTR

19: CLK

40: VCC

39: AD15

38-35: A16-19, S3-S6

34: $\overline{BHE}$, S7

33: MN

ADDRESS AND DATA BUS (AD_0-AD_{15})

PIN #: 2-16, 39
Type: Input / Output

ALE = 1	Contains memory address
ALE = 0	Contains Data

ADDRESS AND STATUS LINES (A16/S3 - A19/S5)

PIN #: 35 - 38
Type: Output

ALE = 1	Contains last 4 bits of memory address and $\overline{BHE}$
ALE = 0	Contains Status bus bits (S_3 - S5, S7)

READ ($\overline{RD}$)

PIN #: 32 Type: Output

0	reading from memory or I/O
1	not reading

READY

PIN #: 22 Type: Input

0	will force the microprocessor to remain idle (wait state)
1	no effect on microprocessor

INTR

PIN #: 18
Type: Input

0	hardware isn't interrupted
1	honored only when IF (interrupt flag) = 1 and $\overline{INTA}$ = 0

CLK

PIN #: 19
Type: Input

33% 67% 33% 67%

RESET

PIN #: 21
Type: Input

1	CS = FFFFH executing address IP = 0000H starts @ FFFF0H
0	NO RESET

MIN MODE PINS

M/$\overline{IO}$
PIN #: 28
Type: output

0	reading from I/O
1	reading from memory

WRITE ($\overline{WR}$)
PIN #: 29
Type: output

0	is writing / contains valied for memory / I/O (databus)
1	not writing data

$\overline{INTA}$
PIN #: 24
Type: output

is 0 when hardware is interrupted

ALE
PIN #: 25
Type: input

when 0 data bus contains data
when 1 contains addresses

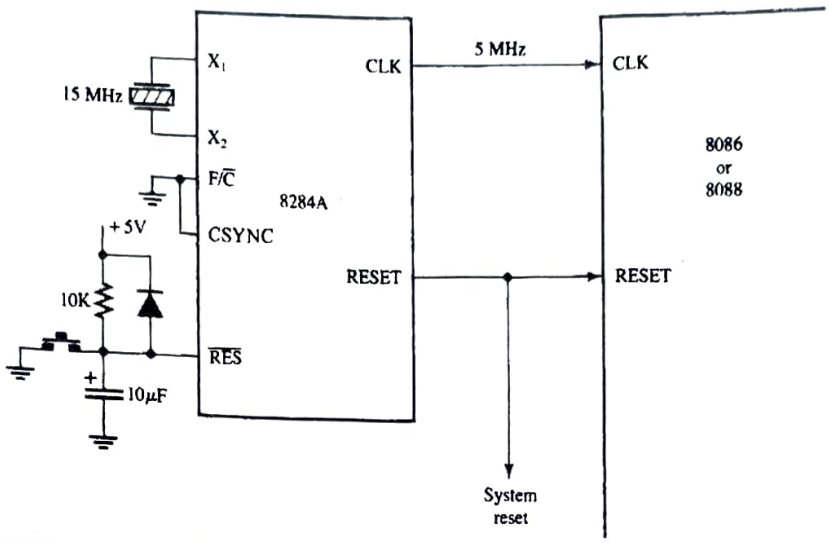
DT/$\overline{R}$
PIN #: 27
Type: output

0	databus recieves data
1	transmit/output data

DEN (data bus enable)
PIN #: 26
Type: output

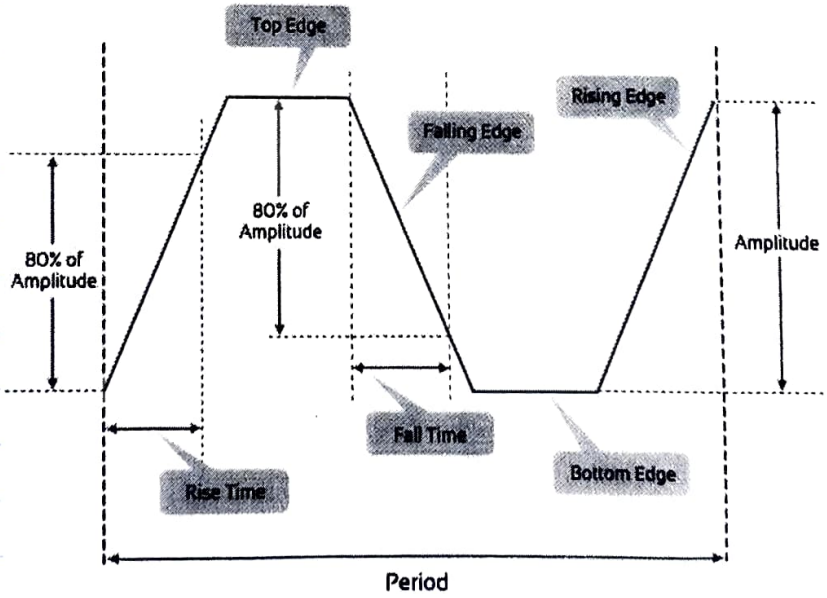
is 0 when AD bus carries data

CLOCK Generator 8284A

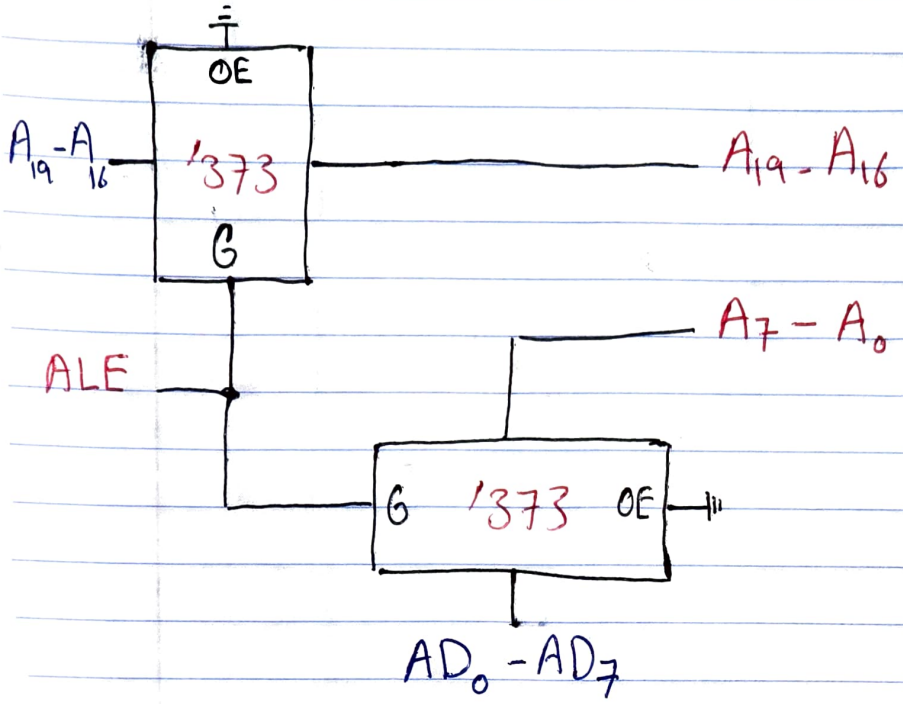


REVISION OF CLOCK WAVE

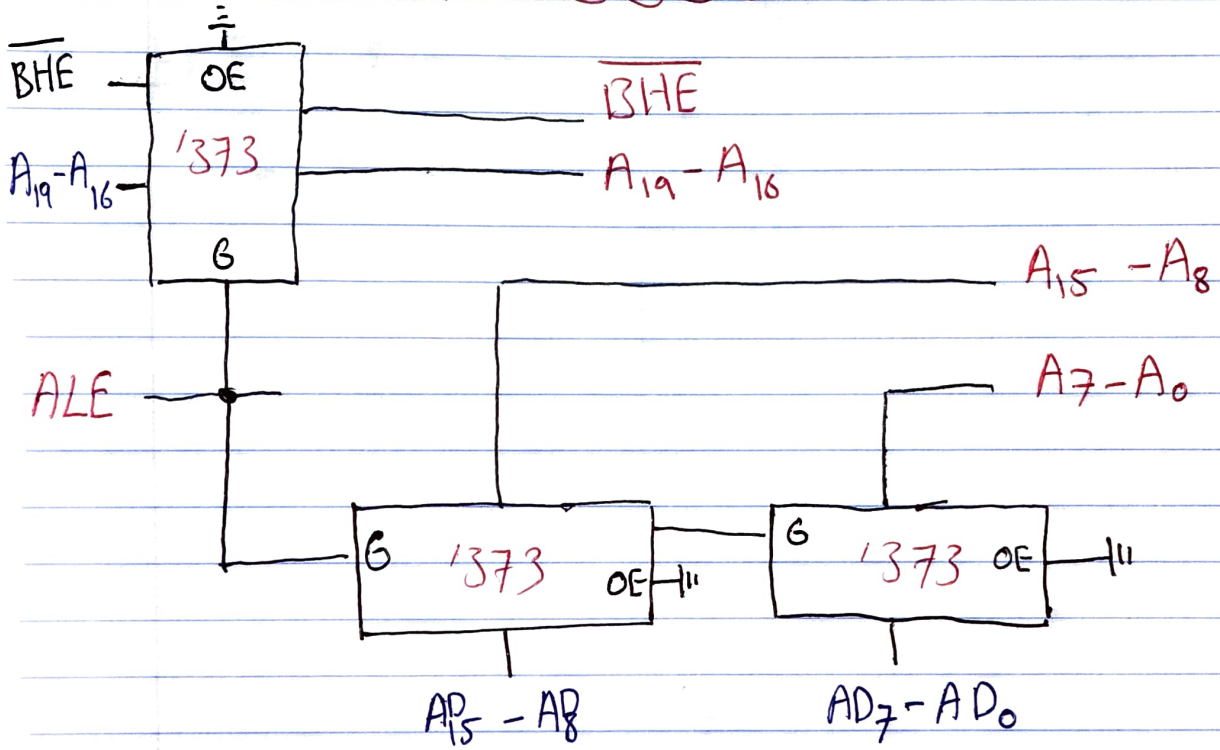
→ for personal Knowledge
- للمعرفة -



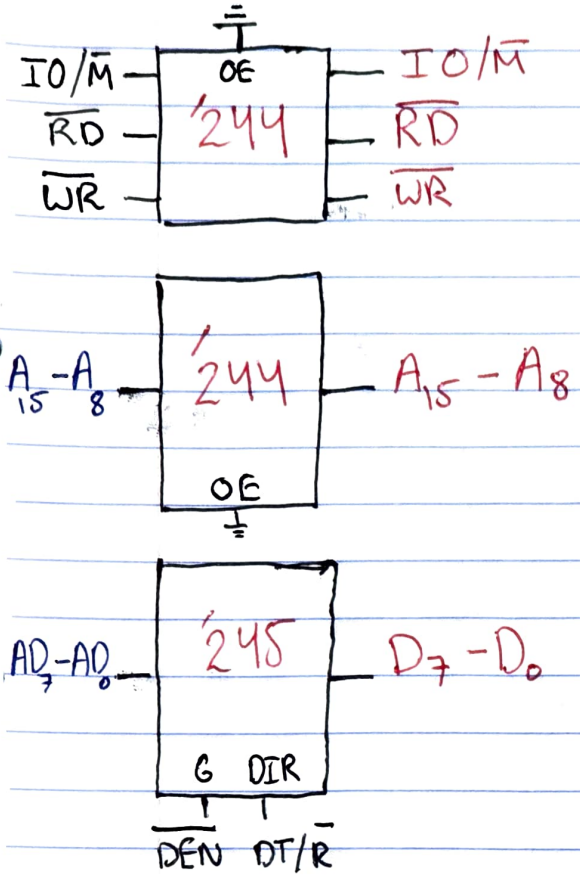
DEMULTIPLEXING 8088



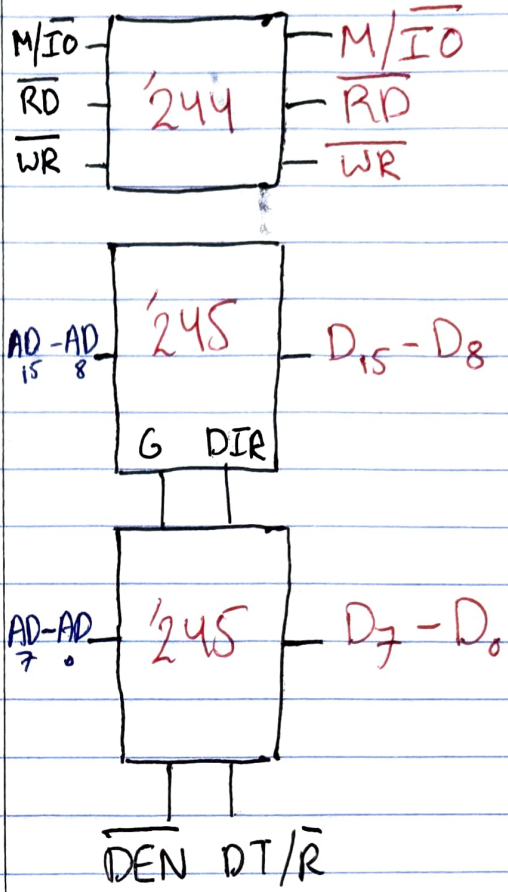
DEMULTIPLEXING 8086



BUFFERING 8088



BUFFERING 8086



GENERAL BUS TIMING

T1 A valid address is placed on address bus (A_0-A_{15} & A_{16}-A_{19})
ALE, DT/$\overline{R}$, IO/$\overline{M}$ are generated

T2 Status is placed on bus instead of address (S_3-S_6)
$\overline{DEN}$, $\overline{RD}$ are issued | if write operation then data appears
READY is issued at the end | if read operation then data appears in T_3 after float

T3 if read operation then Data appears on AD (D_0-D_{15})

T4 ALL SIGNALS ARE RESET TO INITIAL STATE

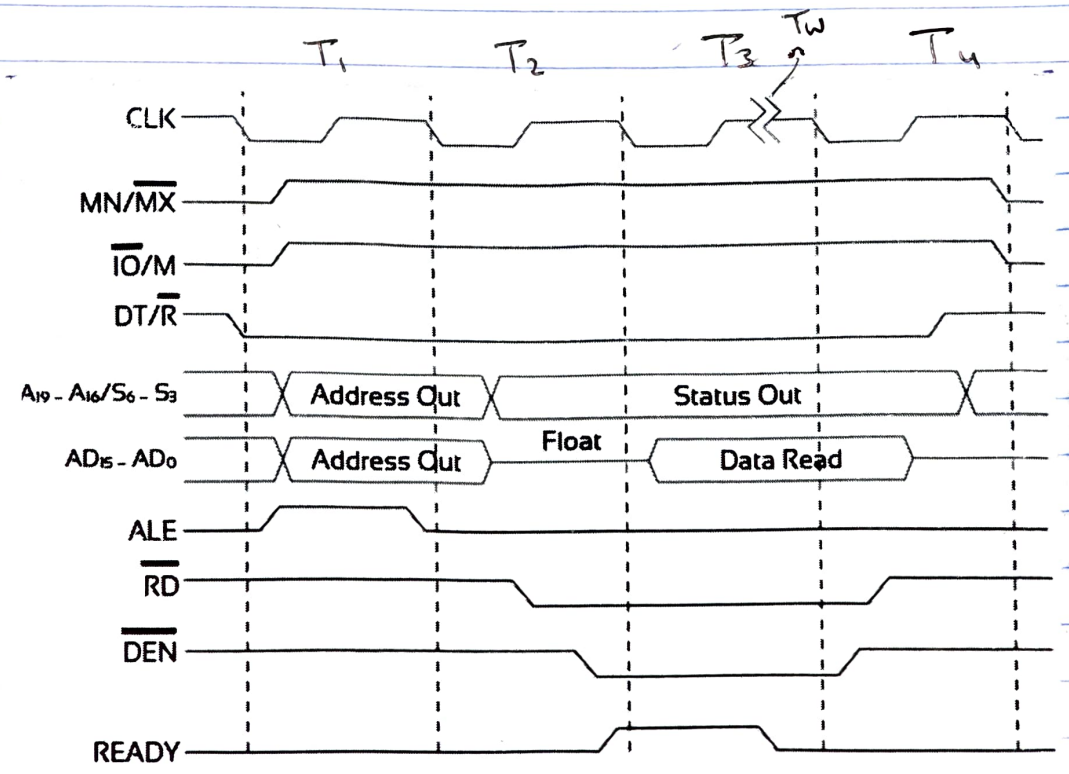


Diagram for memory read
8086

HOW TO SOLVE BUS TIMING QUESTION

→ Step ① find $T = \frac{1}{f}$

→ Step ② find T_{access} =

$$T_{access} = 3T - T_{CLAV} - T_{setup}$$

→ Step ③ find physical address by adding logical addresses of register and offset register (ex: DS + (from address))

note* : access time of RAM given in a question is the float/wait time

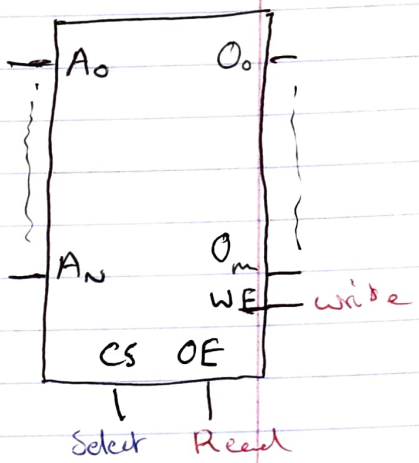
CH 10

MEMORY INTERFACE

1K memory ⟶ 2^{10} $(A_0 - A_9)$

4K memory ⟶ 2^{12} $(A_0 - A_{11})$

8-bit memory is called byte-wide



CONNECTIONS

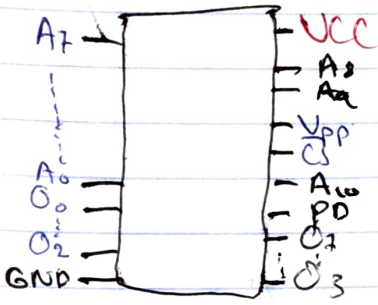
CS/CE (chip select, chip enable)	selects / enables memory. if more than one is present all must be activated
OE/G (output enable gate)	: enables / disables read. found in ROM
WE/(R/W)	: enables / disables write. found in RAM

ROM (Read only memory)

ROM	programed @ factory
PROM	programmed once
EPROM	programmed multiple times (by UV light for 20 min)
EEPROM	(electronically in system used in Bios, cameras, mp3, USB

EPROM (27XX)

27N : N indicates number of bits (2704, 2708, 2716 ... etc.)



→ O : outputs

→ $\overline{CS}$: chip select

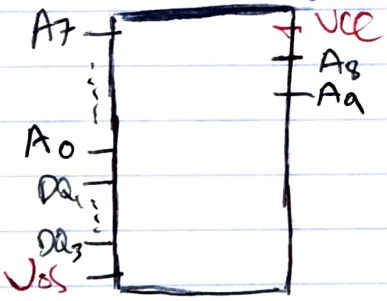
→ PD/PGM : power down / program

→ $A_0 - A_{10}$: Addresses

RAM

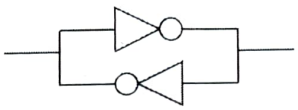
better Read/Write , volatile (disappears power off)

→ SRAM uses flip-flops, fast, used as cache



→ DRAM Dynamic, simple, slower, used as main memory

SRAM VS DRAM

	Static Random Access Memory (SRAM)	Dynamic Random Access Memory (DRAM)
Storage element		
Advantages	1. Fast 2. No refreshing operations	1. High density and less expensive
Disadvantages	1. Large silicon area 2. expensive	1. Slow 2. Require refreshing operations
Applications	High speed memory applications, Such as cache	Main memories in computer systems

ADDRESS DECODING

ADDRESS DECODING STEPS

→ ① Determine range of address for each device

→ ② Determine which address lines go to decoder
- find # address lines on device
- find pattern

→ ③ Design decoder to detect required address bus pattern

USING NAND

refer to slide 18/ch 10

① use memory size (KB) to determine bit to be left as is

ex: 8KB → 2^{13} → A_0-A_{12} left as is
A_{13}-A_{19} to be decoded

② if specific address is specified use adequate gates to modify nand

ex: 40000H is to be interfaced then
if decoded Addresses are A_{16}-A_{19} then
this is how they will be "NAND"ed

$$\overline{A_{19}}\,\overline{A_{18}}\,\overline{A_{17}}\,\overline{A_{16}}$$

USING DECODER

refer to slide 28/Ch10

① use memory size (KB) to determine bits to be left as is and which to be used

ex: 8KB → 2^{13} → A_0-A_{12} left as is
A_{13}-A_{19} to be decoded

② based off # chips determine which bits are to be used as input (from to be decoded)

ex: # chips 4 → first two are input rest are NAND as reset
chips 8 → first three " " " " " " "

③ if specific address is specified use adequate gates to modify reset/NAND

CH11

I/O INTERFACE

IN, INS, OUT, OUTS

IN/OUT : between reg and port

INS/OUTS : between port in DX & DI

ISOLATED I/O & MEMORY

→ Isolated I/O uses IN/INS/OUT/OUTS and has it's own adress space

→ MEMORY mapped I/O uses MOV (address shared between memory & I/O)

PC I/O MAP

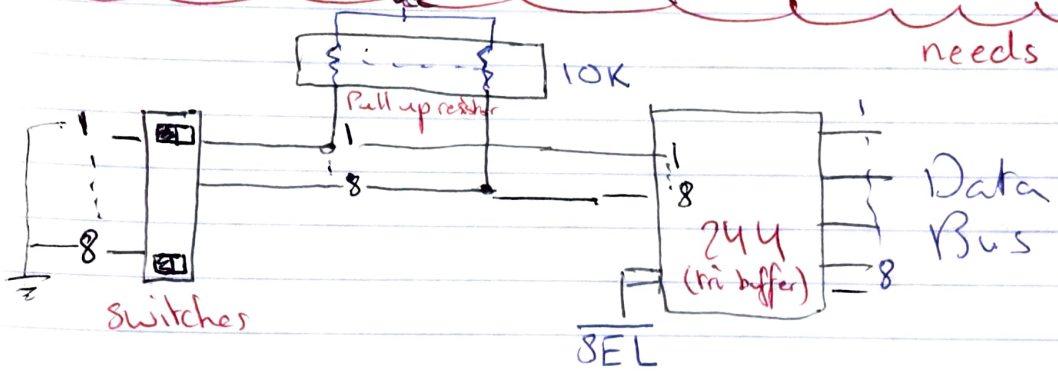
0000H - 03FFH : system & ISA bus

0400H - FFFFH : pci bus & user applications

slides 1-7

BASIC 8-BIT INPUT PORT INTERFACE

needs tri-state buffer



SWITCHES

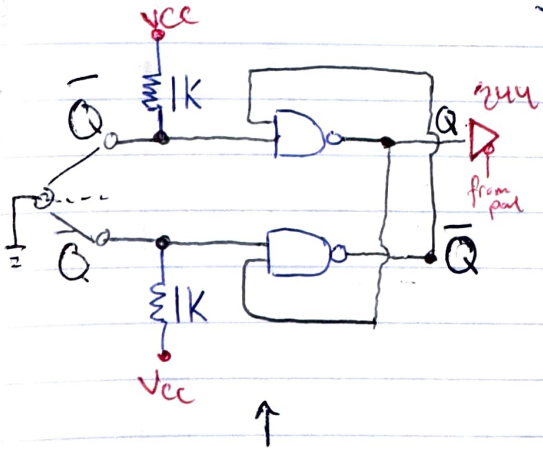
→ Don't produce voltage

→ To make TTL compatible pullup resistor (1K-27K) is used

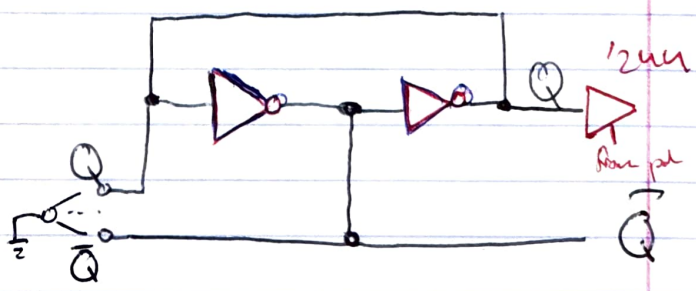
→ Their main problem is Bouncing. can be solved using software (HTL) or Hardware using 244 Buffer

SWITCH DEBOUNCING

→ SR Debouncers



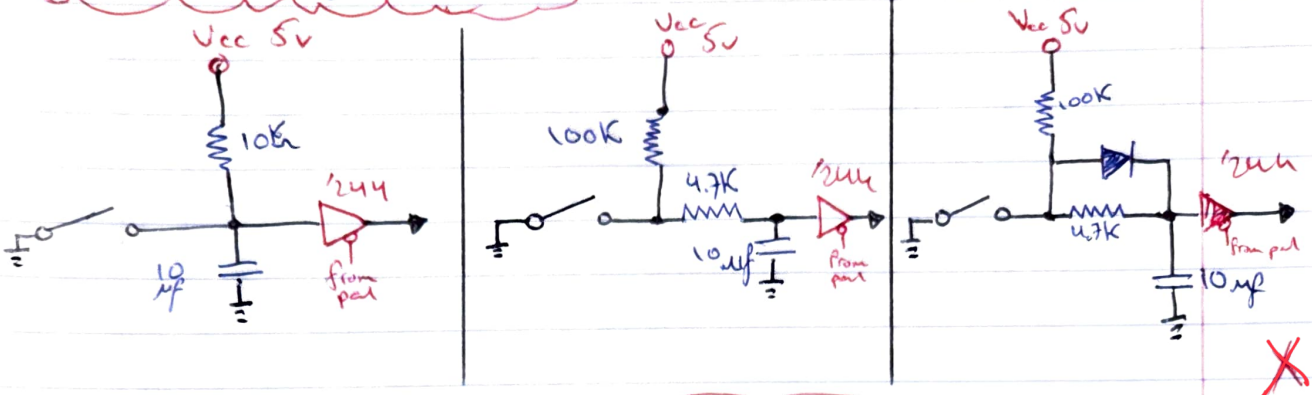
USING NAND



USING INVERTER

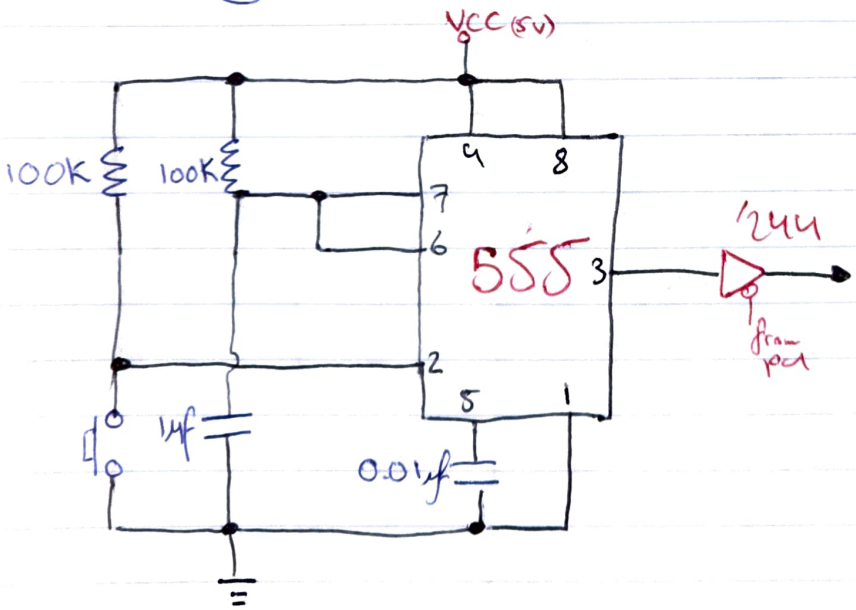
- no pull up resistor used -

RC DEBOUNCER



MONO-STABLE/MULTI-VIBRATORS DEBOUNCER

(Debouncing using 555 - timer)



refer to (slide 19 / ch 11) Part 2 ex

READ DEBOUNCED USING ASSEMBLY (4 buttons)

OPEN:

```
IN AL, PORT
AND AL, OFH
CMP AL, OFH
JNZ OPEN
CALL DELAY_20MS
```

CLOSE:

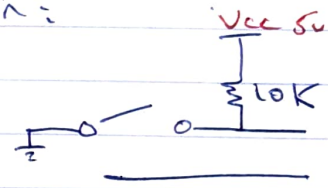
```
IN AL, PORT
AND AL, OFH
CMP AL, OFH
JZ CLOSE
CALL DELAY_20MS
IN AL, PORT
AND AL, OFH
CMP AL, OFH
JZ CLOSE
```

```
CMP AL, 0EH , first button
JE ACTION1
CMP AL, 0DH , 2nd button
JE ACTION2
CMP AL, 0BH , 3rd button
JE ACTION 3
CMP AL, 07H , 4th button
JEACTION 4
JMP OPEN
```

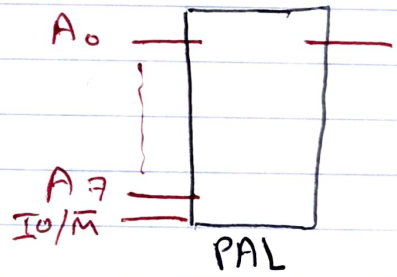
(8088)

ex: draw interface for software debounced

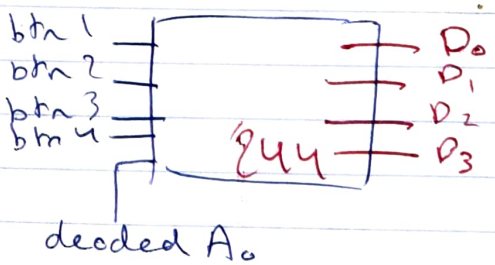
btn:



Address decoding



interface:



VIP

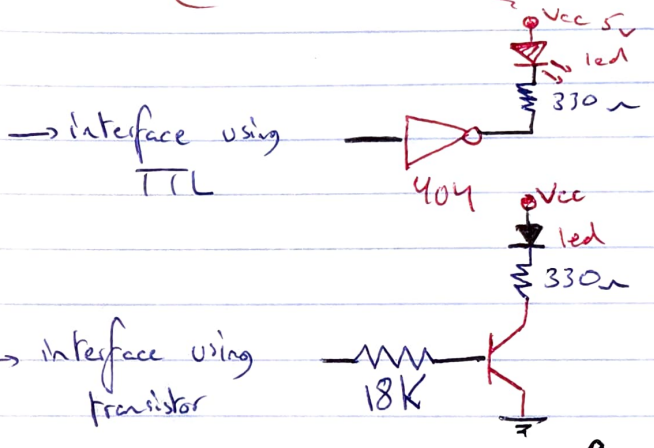
8-BIT OUTPUT

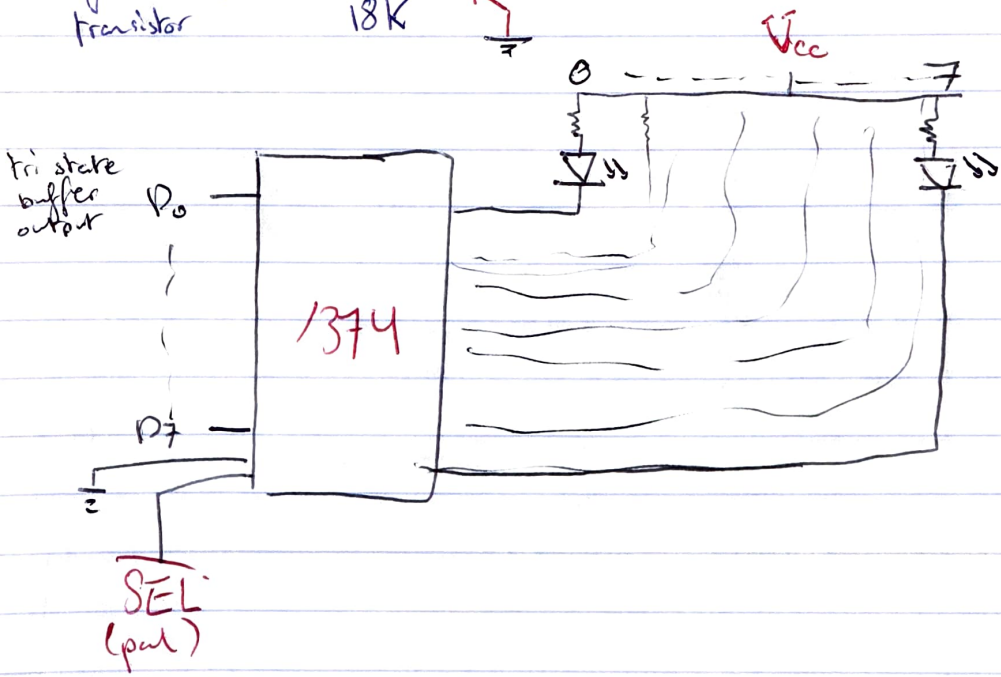
→ microprocessor

	voltage	current
logic 0	= 0 – 0.4 V	logic 0 = 0 – 2 mA
logic 1	= 2.4 – 5 V	logic 1 = 0 – 400 μA

→ for LED (RGB)

10mA , V_{diode} = 1.65 V

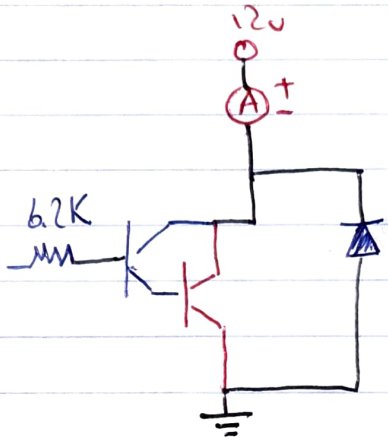




DC MOTORS

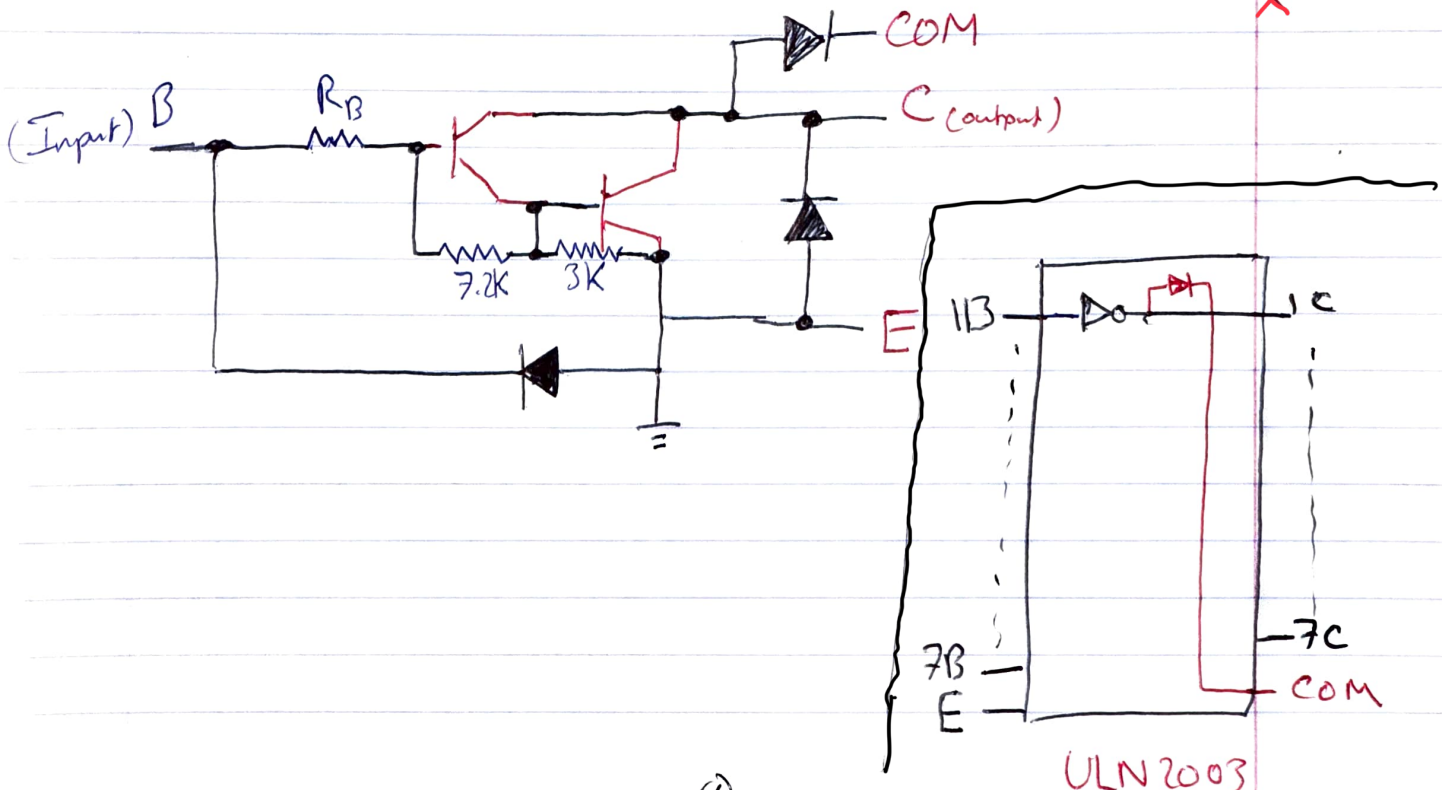
→ use 12 v or higher power supply & 1A

→ TTL can't be used instead we use Darlington pair



$\beta = \beta_1 \beta_2$

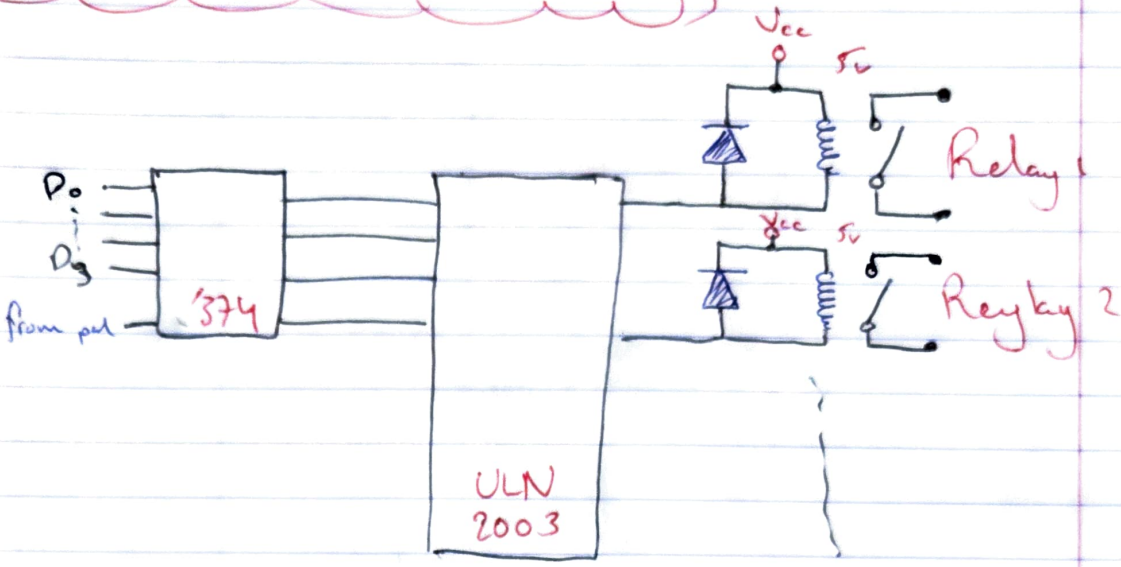
RELAYS



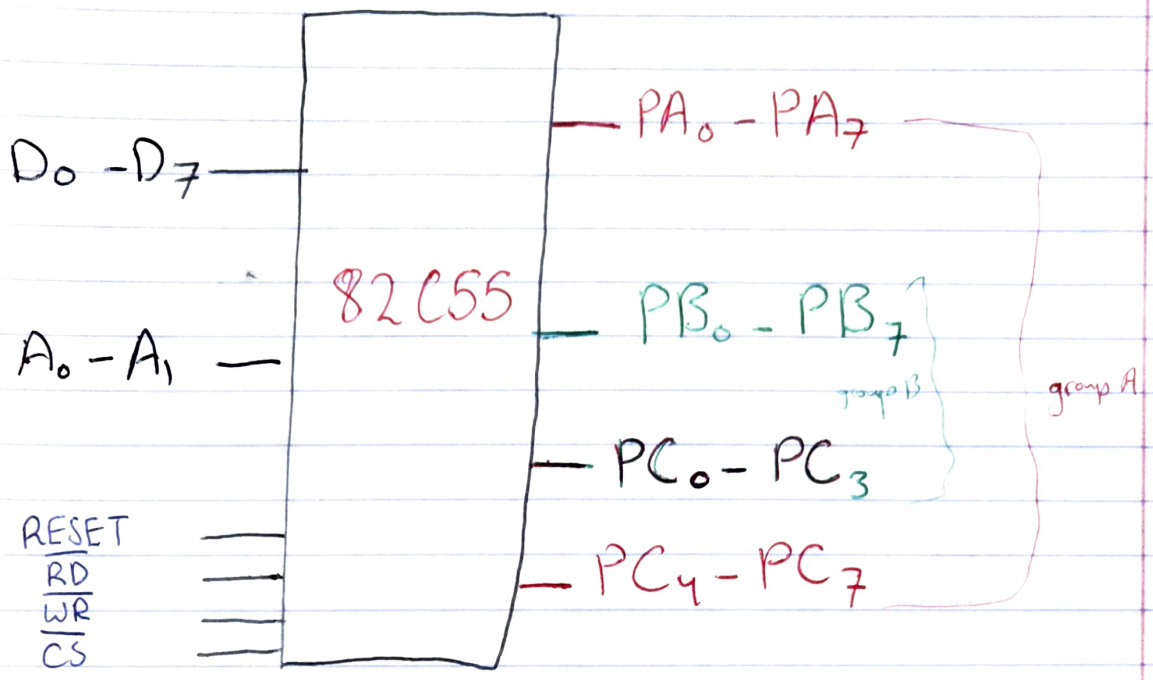
STEPPER MOTOR CODE

```
MOV BL, 11      ; initialize

AGAIN:
        IN AL, BTN_ADD
        AND AL, 3
        CMP AL, 2
        JE btn0
        CMP AL, 1
        JE btn1
        CALL DELAY
        JMP AGAIN

btn0: ; rotate clkwise 90°
      MOV CX, 50 ---> ; based off calculated steps
       LP:
            MOV AL, BL
            OUT 0FFH, AL
            ROR BL, 1
            CALL DELAY
       LOOP LP
       JMP AGAIN

btn1: ; rotate cc 180°
      MOV CX, 100
       LP1:
            MOV AL, BL
            OUT 0FFH, AL
            ROL BL, 1
            CALL DELAY
       LOOP LP1
      JMP AGAIN

DELAY PROC
    PUSH CX
    MOV CX, 0FFH
     LLP: nop
     LOOP LLP
     POP CX
DELAY ENDP
```

FULL RELAY INTERFACE CCT



82C55 (PPI)



→ A_0, A_1 are control lines and are connected to PAL output

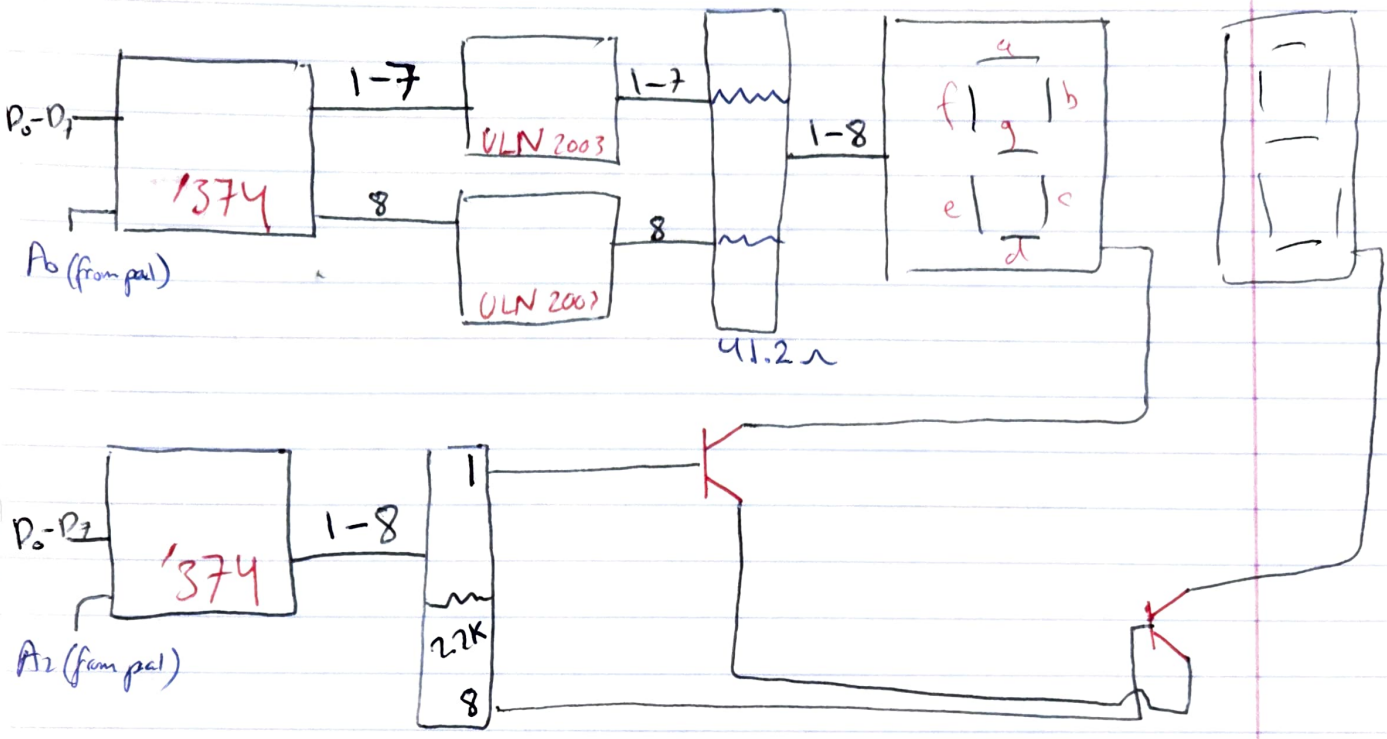
- 0 0 Port A
- 0 1 Port B
- 1 0 Port C
- 1 1 CGW

PPI PROGRAMING

programmed using internal register (COMMAND PORT/BYTE)

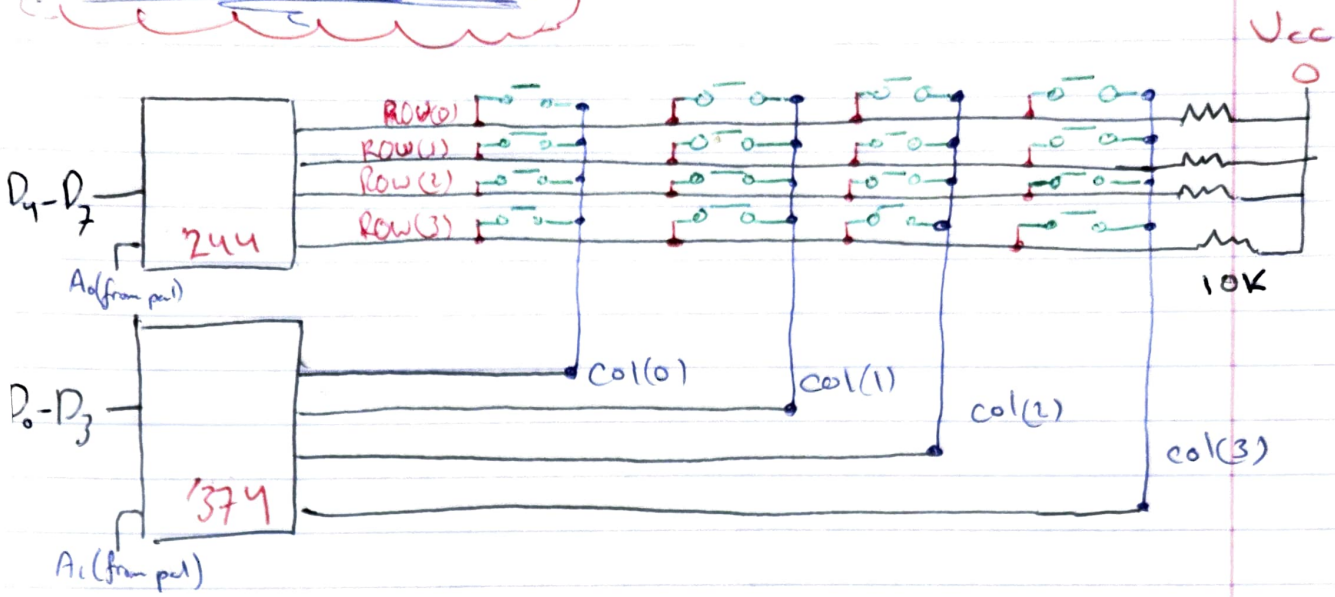
	7	6	5	4	3	2	1	0	Bit
group A →	1: command byte A	A_1	A_0	X	X	0	X	X	Value
group B →	0: byte B								

SEVEN SEGMENT DISPLAY




```
MOV SI, OFFSET DISP
MOV AH, 7FH
MOV CX, 8        ; 8 Digits

LP:
    MOV AL,AH
    OUT PORT,AL
    MOV AL [SI]
    OUT PORT AL
   CALL DELAY
   ROR AH, 1
   INC SI
   LOOP LP
```

KEYPAD MATRIX



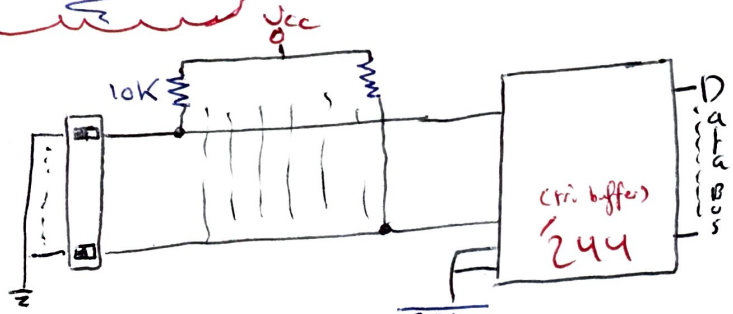
→ code below is to read the top row of a 3×3 keypad

activate
↓ ↑ ↓
MOV AL, 30h ; for 1st column 00110000
OUT C_ADD, AL
IN AL, R_ADD
TEST AL, 1
JE PRESSED1
jmp nextCOL
pressed1:

D_0	1	2	3
D_1	4	5	6
D_2	7	8	9
	D_3	D_4	D_5

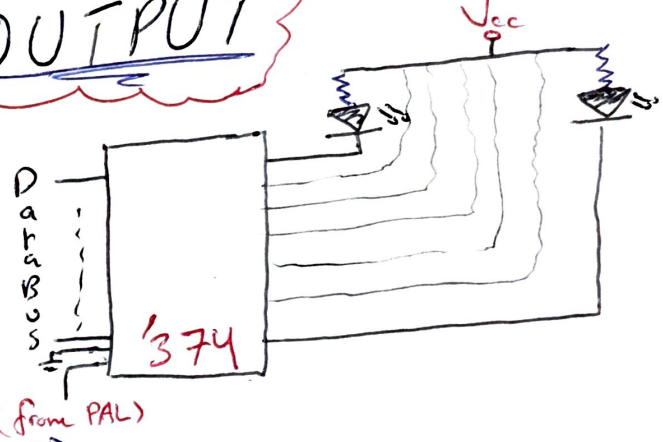
nextCOL: 28h
; same as 1st part but for 2nd column 00101000

INPUT

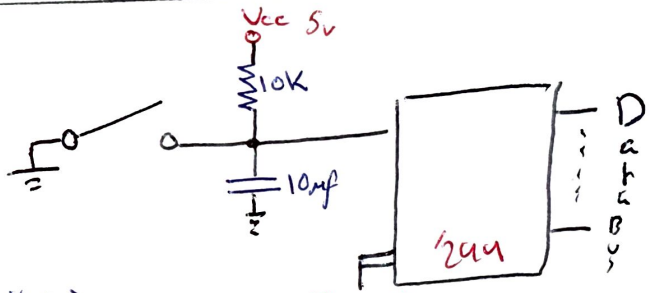


(Slider switches)

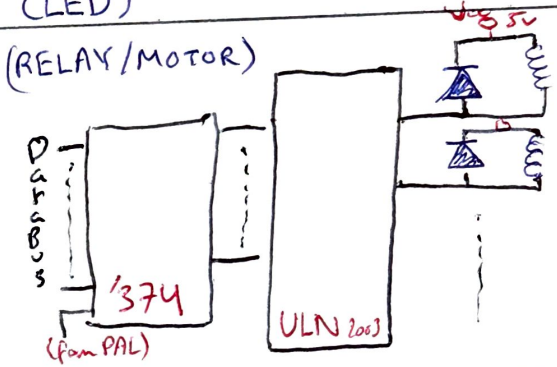
OUTPUT

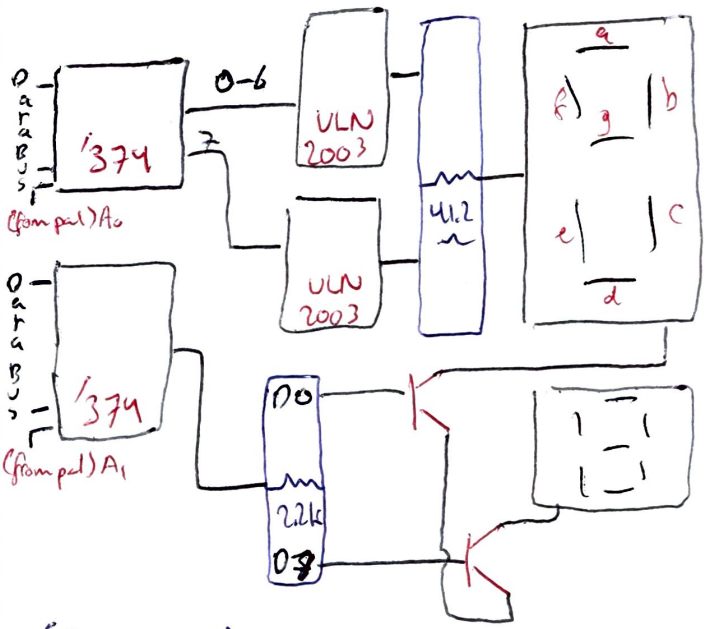


(LED)



(switch)





(7 segment)