LAPORAN RESMI
PRAKTIKUM 3 ARSITEKTUR KOMPUTER

“SWITCH/PUSH BUTTON INPUT”

Disusun Oleh :
Izzuddin Ahmad Afif (2421600011)

Dosen :
Mohamad Ridwan S.T., M.T.

PROGRAM STUDI SARJANA TERAPAN TEKNOLOGI REKAYASA INTERNET
DEPARTEMEN TEKNIK ELEKTRO
POLITEKNIK ELEKTRONIKA NEGERI SURABAYA

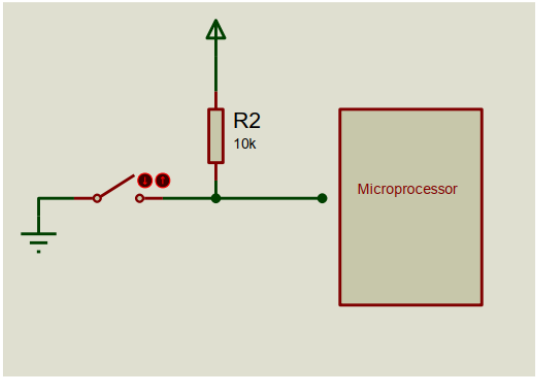
2021/2022

BAB I
PENDAHULUAN

1. Dasar Teori

Switch atau Push Button adalah perangkat input dasar dalam embedded system yang terlihat dalam sistem yang sangat sederhana hingga yang sangat kompleks. Mereka adalah tombol on-off mekanis dasar yang bertindak sebagai perangkat kontrol. Ketika switch ditekan kan mengindikasikan logic 1 / 0 atau sebaliknya. Hal ini yang dapat dibaca oleh sebuah mikroprosesor atau mikrokontroler sehingga dapat mengeksekusi perintah yang diberikan.

Secara umum, sakelar diklasifikasikan menjadi 2, sakelar mekanis dan sakelar listrik/elektronik. Sakelar mekanis diklasifikasikan menjadi lima pada dasarnya, yaitu switch SPST (Single Pole Single Throw), SPDT (Single Pole Double Throw), DPST (Double Pole Single Throw), DPDT (Double Pole Double Throw) dan 2P6T (2 Poles 6 Throw). Perangkat elektronik seperti transistor, MOSFET, dan relai dapat bertindak sebagai sakelar dan termasuk dalam kategori sakelar listrik/elektronik.



Gambar 1. Rangkaian interface dasar mikroprosesor dengan switch

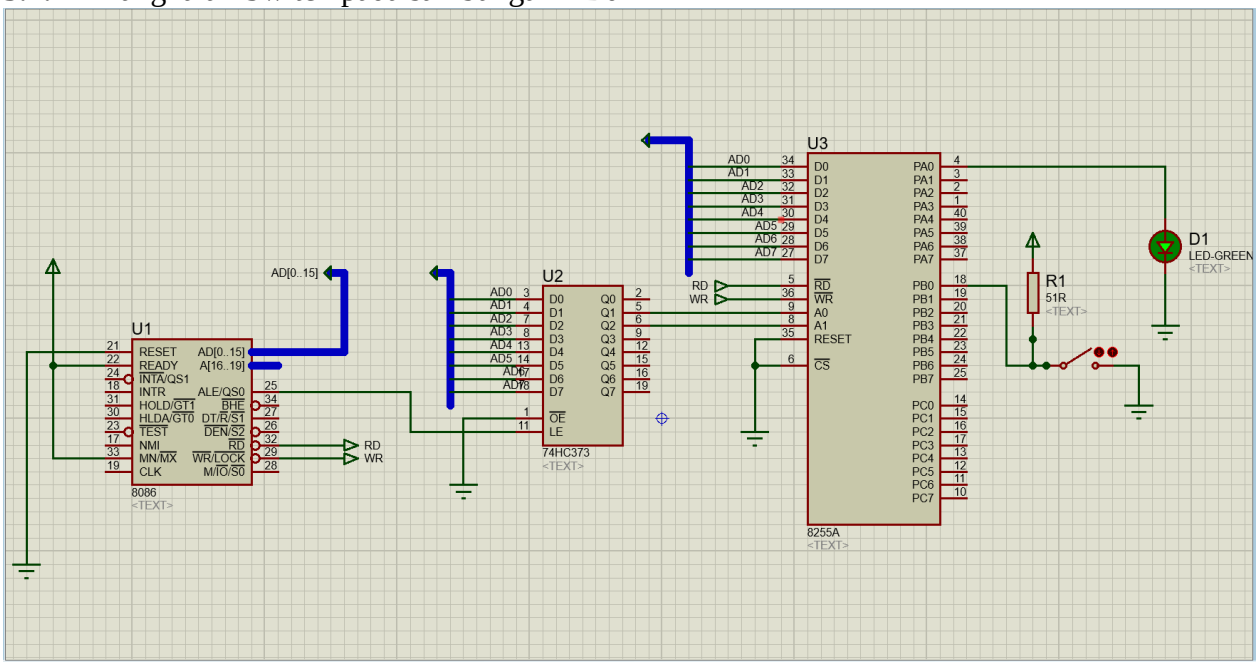
BAB II
METODOLOGI PRAKTIKUM

1. **Alat atau Bahan Praktikum**
   - **a.** Proteus Professional : https://downloadly.net/2020/13/3175/03/proteus/03/?#/3175-proteus-032127081430.html
   - **b.** EMU8086 : https://drive.google.com/drive/folders/1OPVhsYiHJm3_rfvUWiqL9yJW5Wn7S3LU
   - **c.** Datasheet Intel 8086
   - **d.** Instruction set for Intel 8086
   - **e.** Datasheet IC 8255 PPI
   - **f.** Datasheet IC 74HC373

BAB III
ANALISA DAN KESIMPULAN

3.1 Hasil Praktikum

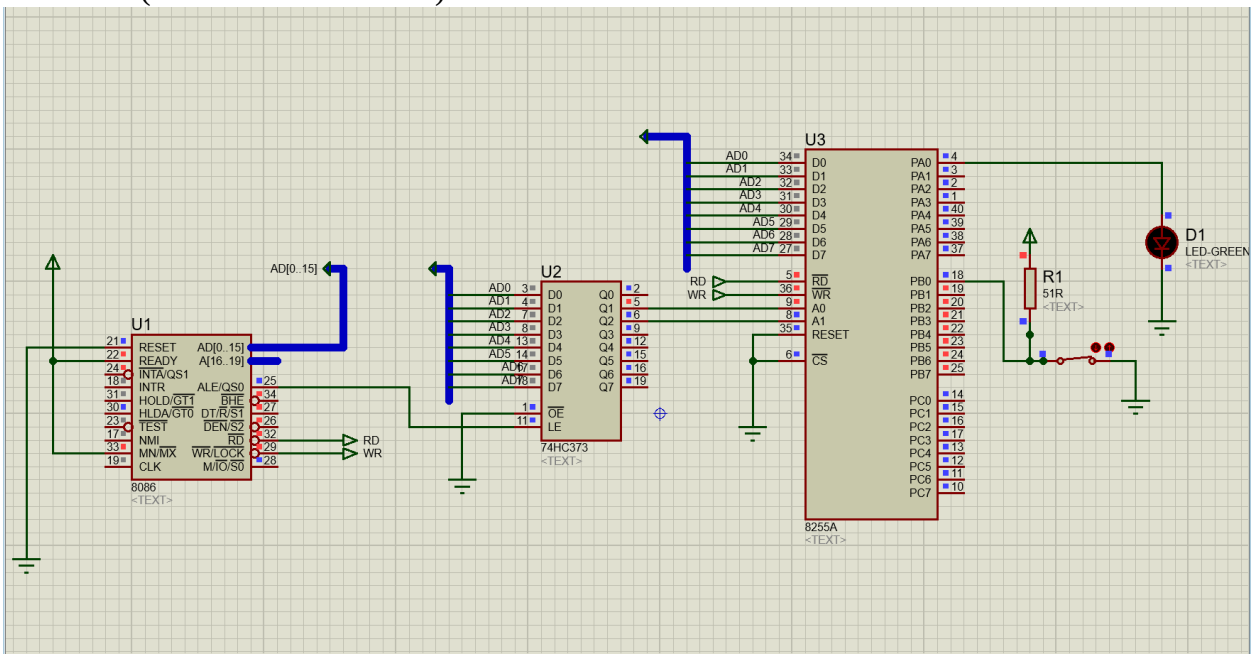
3.1.1 Rangkaian Switch pada sambungan **PB0**



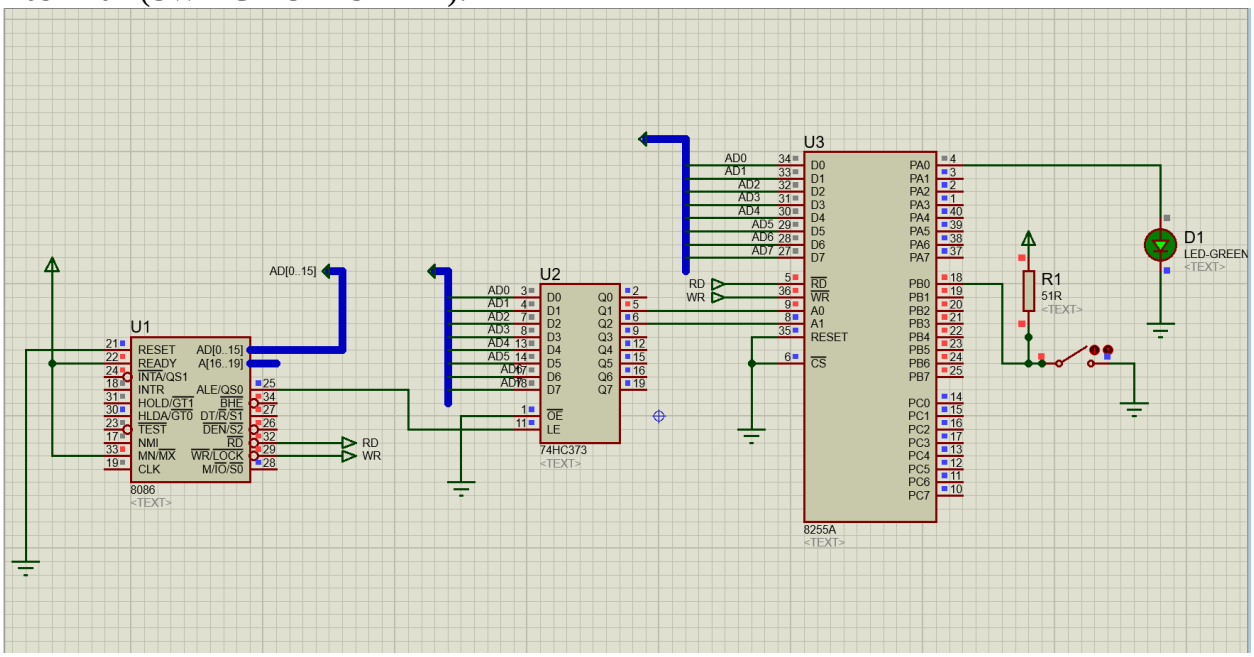
Source Assembly code beserta penjelasan:

```
;add your code here
START:

MOV DX, PORT_CON; moves control port address to dx register
MOV AL, 10000010B; sets control word
OUT DX, AL; copies bytes of control word in AL to control port addressed by dx

JMP X1; unconditional jump to x1

X1:; Checking input from certain port address.
IN AL, PORTB; copies port size value/addresses of certain port to AL
AND AL, 01H; performs bitewise AND on immediate value with AL's value,
           ; stores bool val. result in A, checking if such address does exist in
           ; certain port and whether there is any input from the address.
CMP AL, 01H; compares immediate bool value with AL's bool value (temp=source1-signextend(source2)).
           ; sets ZF status flag to ZF=0 or ZF=1 (bool val.) based on condition tested for,
           ; either Zero or Not Zero. if there's input from certain address (temp==0), then ZF=1 vice versa.
JNZ XON; conditional jump to XON if ZF==1
JZ XOFF; conditional jump to XOFF if ZF==0

XON:; SWITCH ON STATE
CALL delay_20ms ;debounce
MOV AL, 0000H; 0H addresses nothing in port A, so when switch is on,
MOV DX, PORTA; output is off.
OUT DX, AL
JMP X1; loop to keep the LED on.

XOFF:; SWITCH OFF STATE
CALL delay_20ms ;debounce
MOV AL, 0001H; When switch is off, output at 01H is on.
MOV DX, PORTA
OUT DX, AL
JMP X1; loop to keep the LED off

;Delay of 20ms
delay_20ms PROC near
MOV     CX, 2220H
x9: LOOP    x9
RET
delay_20ms  ENDP

CODE ENDS
END

HLT             ; halt!
```

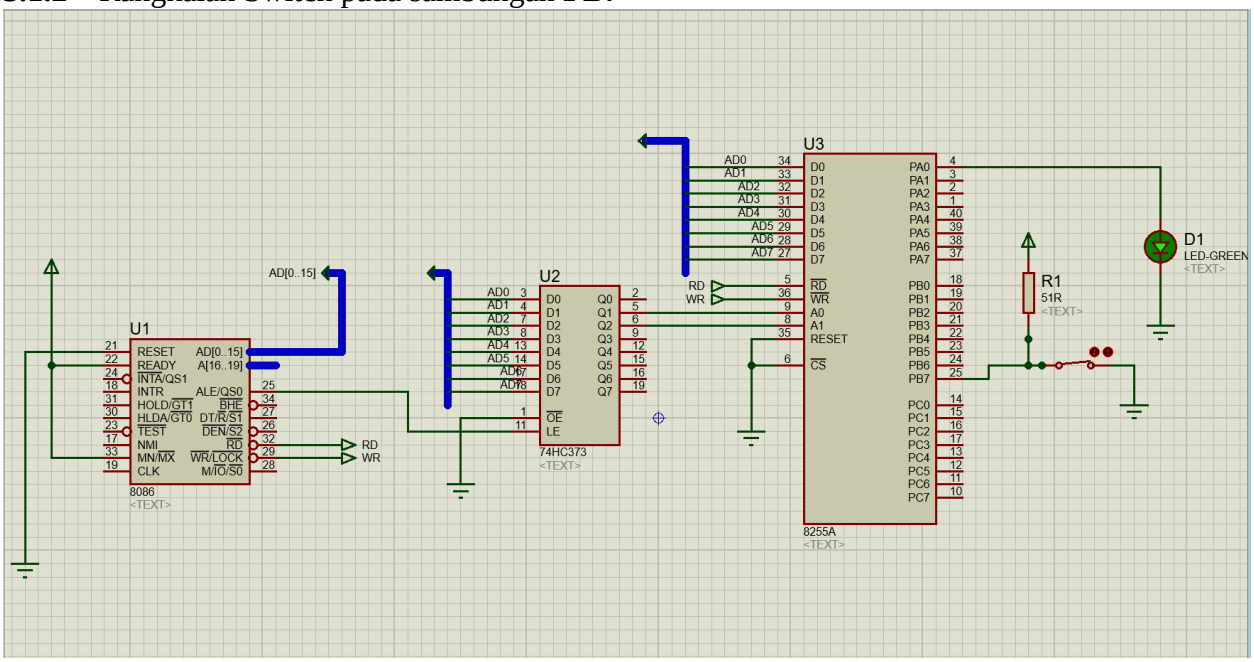
Hasil Run (SWITCH ON STATE):



Hasil Run (SWITCH OFF STATE):



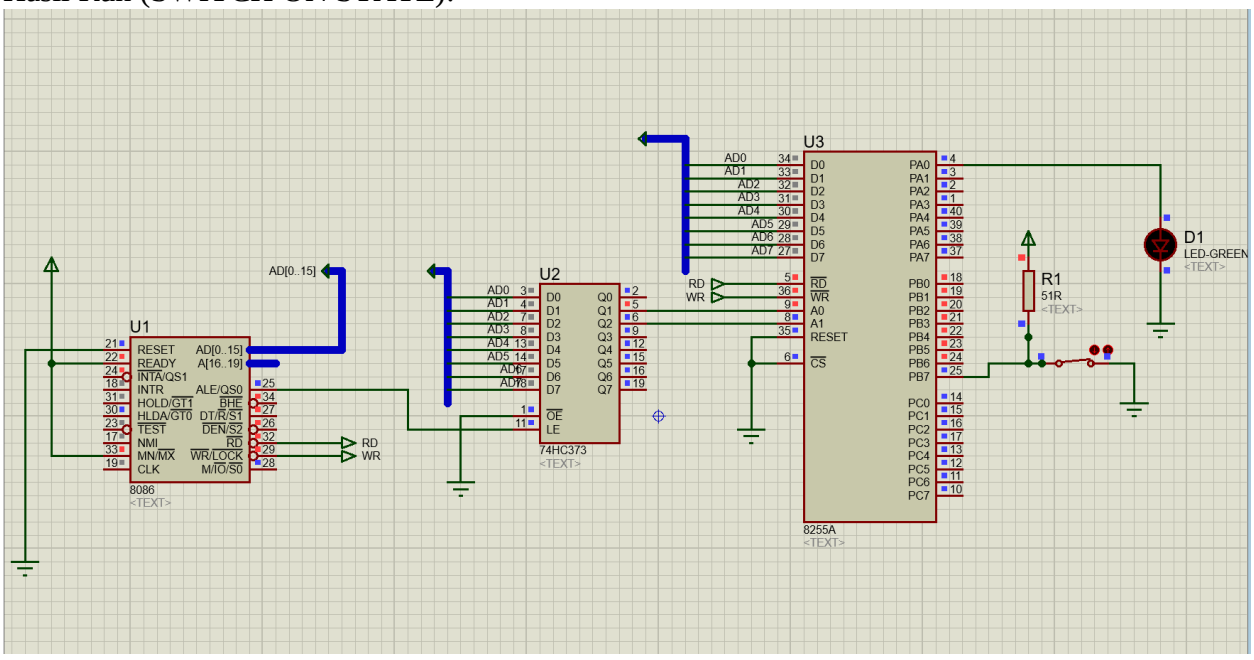
3.1.2 Rangkaian Switch pada sambungan **PB7**



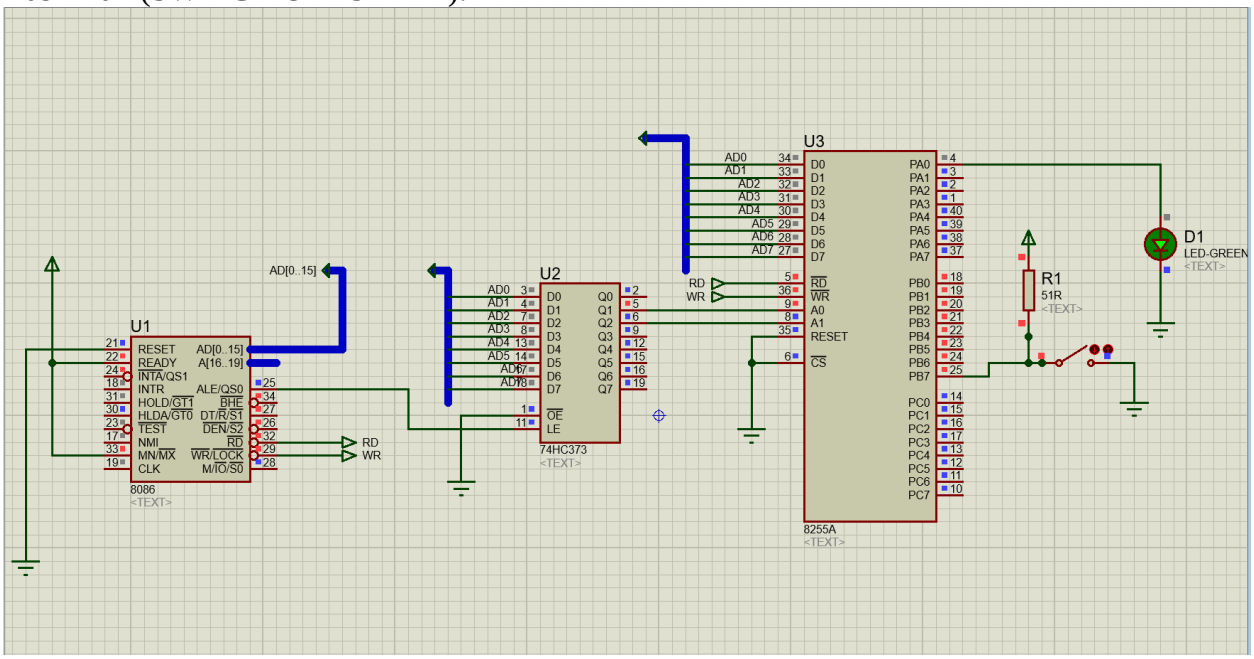
Source Assembly Code:

```
;add your code here
START:

MOV DX, PORT_CON; moves control port address to dx register
MOV AL, 10000010B; sets control word
OUT DX, AL; copies bytes of control word in AL to control port addressed by dx

JMP X1; unconditional jump to x1

X1:; Checking input from certain port address.
IN AL, PORTB; copies port size value/addresses of certain port to AL
AND AL, 80H; performs bitewise AND on immediate value with AL's value,
          ; stores bool val. result in A, checking if such address does exist in
          ; certain port and whether there is any input from the address.
CMP AL, 80H; compares immediate bool value with AL's bool value (temp=source1-signextend(source2)).
          ; sets ZF status flag to ZF=0 or ZF=1 (bool val.) based on condition tested for,
          ; either Zero or Not Zero. if there's input from certain address (temp==0), then ZF=1 vice versa.
JNZ XON; conditional jump to XON if ZF==1
JZ XOFF; conditional jump to XOFF if ZF==0

XON:; SWITCH ON STATE
CALL delay_20ms ;debounce
MOV AL, 0000H; 0H addresses nothing in port A, so when switch is on,
MOV DX, PORTA; output is off.
OUT DX, AL
JMP X1; loop to keep the LED on.

XOFF:; SWITCH OFF STATE
CALL delay_20ms ;debounce
MOV AL, 0001H; When switch is off, output at 01H is on.
MOV DX, PORTA
OUT DX, AL
JMP X1; loop to keep the LED off

;Delay of 20ms
delay_20ms PROC near
MOV     CX, 2220H
x9: LOOP     x9
RET
delay_20ms  ENDP

CODE ENDS
END

HLT           ; halt!
```

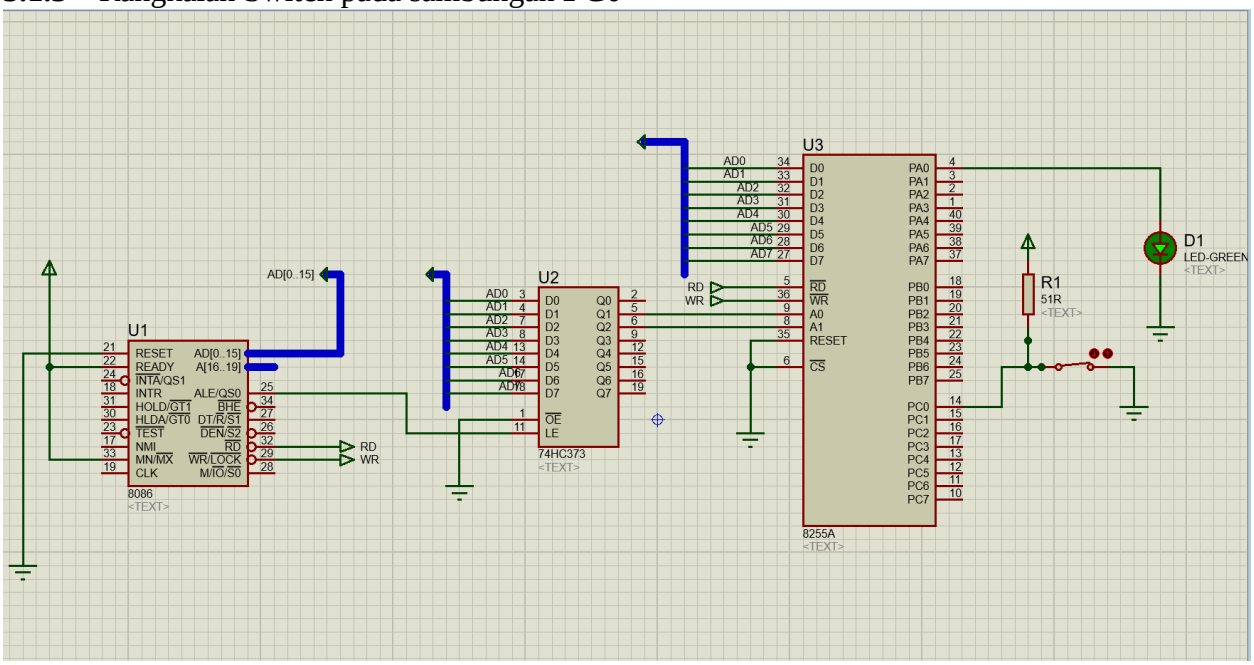
Hasil Run (SWITCH ON STATE):



Hasil Run (SWITCH OFF STATE):



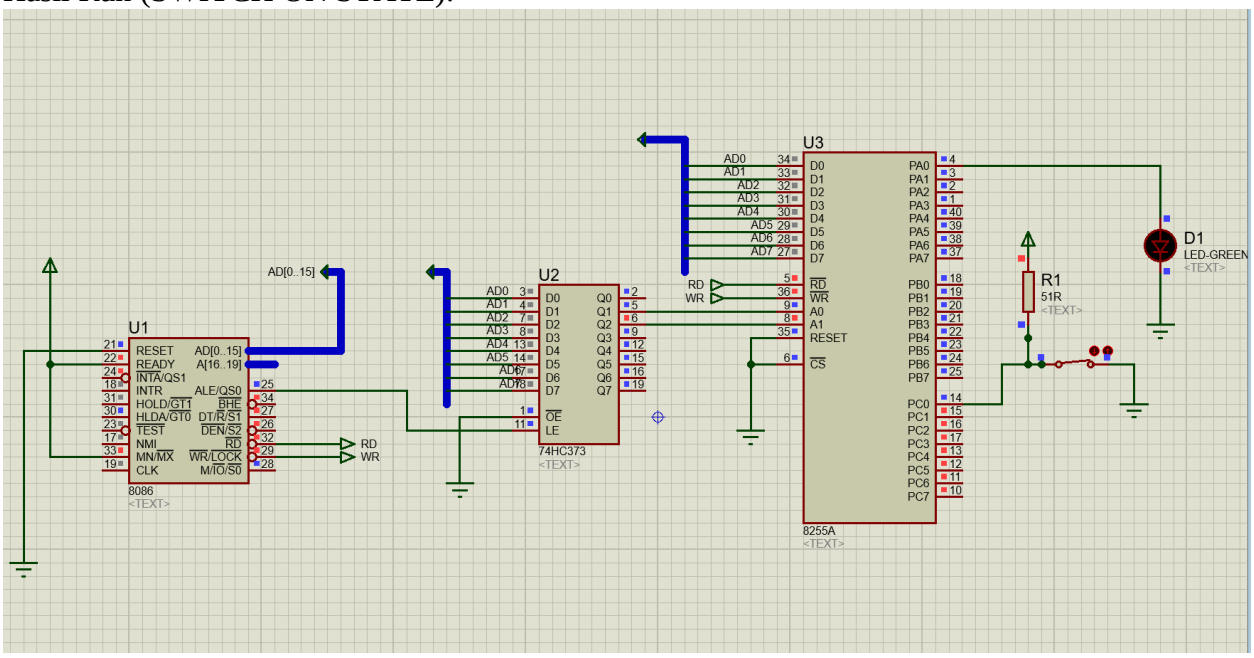
3.1.3 Rangkaian Switch pada sambungan **PC0**



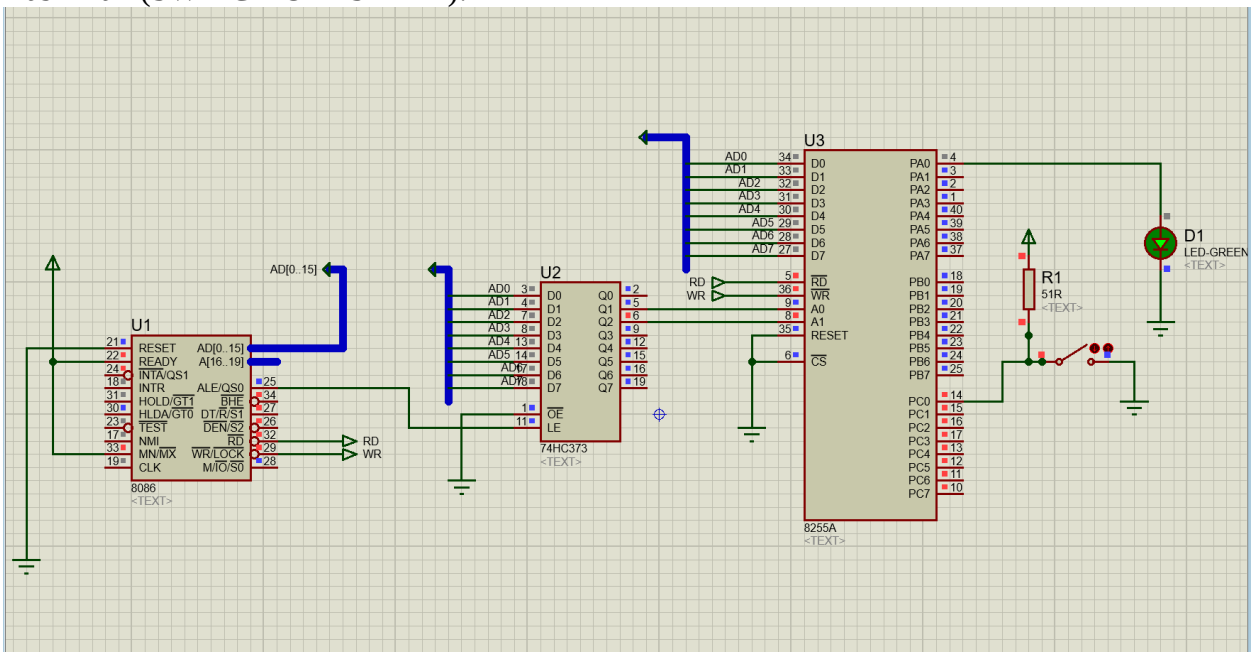
Assembly Source Code:

```
;add your code here
START:

MOV DX, PORT_CON; moves control port address to dx register
MOV AL, 10001001B; sets control word
OUT DX, AL; copies bytes of control word in AL to control port addressed by dx

JMP X1; unconditional jump to x1

X1:; Checking input from certain port address.
IN AL, PORTC; copies port size value/addresses of certain port to AL
AND AL, 01H; performs bitewise AND on immediate value with AL's value,
          ; stores bool val. result in A, checking if such address does exist in
          ; certain port and whether there is any input from the address.
CMP AL, 01H; compares immediate bool value with AL's bool value (temp=source1-signextend(source2)).
          ; sets ZF status flag to ZF=0 or ZF=1 (bool val.) based on condition tested for,
          ; either Zero or Not Zero. if there's input from certain address (temp==0), then ZF=1 vice versa.
JNZ XON; conditional jump to XON if ZF==1
JZ XOFF; conditional jump to XOFF if ZF==0

XON:; SWITCH ON STATE
CALL delay_20ms ;debounce
MOV AL, 0000H; 0H addresses nothing in port A, so when switch is on,
MOV DX, PORTA; output is off.
OUT DX, AL
JMP X1; loop to keep the LED on.

XOFF:; SWITCH OFF STATE
CALL delay_20ms ;debounce
MOV AL, 0001H; When switch is off, output at 01H is on.
MOV DX, PORTA
OUT DX, AL
JMP X1; loop to keep the LED off

;Delay of 20ms
delay_20ms PROC near
MOV     CX, 2220H
x9: LOOP    x9
RET
delay_20ms  ENDP

CODE ENDS
END

HLT           ; halt!
```

Hasil Run (SWITCH ON STATE):



Hasil Run (SWITCH OFF STATE):



3.2 Analisa Praktikum

Dalam Praktikum kali ini, kita akan menguji input yang diberi switch, dengan expected output bila switch off, maka LED on, dan apabila switch on, maka LED off. Prosedur pertama telah dilakukan setelah memasukkan code assembly ke 8086, dan output sesuai harapan.

Prosedur kedua adalah memindah sambungan switch ke PB7, dengan begitu kita mengubah kode assembly (yang kebetulan sudah saya beri keterangan dengan detail dalam bahasa inggris di gambar hasil praktikum) yang mengatur input dari port address tertentu. Kita ganti value

yang diassignkan ke dalam register AL dalam instruksi AND dan CMP menjadi address dari PB7 yaitu 80H atau 10000000B. Sehingga output yang dihasilkan saat run sesuai dengan harapan.

Prosedur Ketiga adalah memindahkan sambungan switch ke PC7. Dalam hal ini, kita tidak hanya mengubah value yang diassignkan ke register AL pada instruksi AND dan CMP menjadi address PC0 (01H) saja, namun kita juga harus mengganti control word yang diassignkan ke Control Port Adress (PORT_CON pada program), yang awalnya 10000010B yang berarti menjadikan Port A dan C sebagai output dan B sebagai input menjadi 10001001B yang berarti menjadikan port A dan B sebagai output dan C sebagai input, lalu kita ganti juga operand dari instruksi IN yang awalnya PORTB menjadi PORTC, untuk memeriksa apakah di PORT C bagian 01H ada input atau tidak. Sehingga dihasilkanlah output sesuai harapan.

3.3 Kesimpulan

Dari praktikum kali ini, kita dapat mengetahui tentang control word, address dari port serta switch input.

TUGAS

1. Ketika menggunakan suatu switch, push button, sebagai data input ke microcontroller terkadang terjadi masalah nilai tidak terbaca. Nilai input tersebut mengambang, float state, antara high dan low. Untuk mengatasi masalah tersebut dapat digunakan resistor pull-up atau pull-down.

Pada dasarnya baik resistor pull-up maupun pull-down, keduanya sama-sama berfungsi untuk menghindari suatu node mengalami nilai yang mengambang, float, antara low dan high.
Kasus pertama, dimana sebuah switch yang satu node terhubung dengan sumber tegangan dan satu node lainnya terhubung dengan pin input microcontroller akan cenderung mengalami masalah kondisi float untuk keadaan low. Pada saat ditekan microcontroller akan menerima data input bernilai high dari switch tersebut, akan tetapi pada saat tidak ditekan nilainya undefined, cenderung float, antara low dengan high.
Kasus kedua, dimana sebuah switch yang satu node terhubung dengan ground dan satu node lainnya terhubung dengan pin input microcontroller akan cenderung mengalami masalah kondisi float untuk keadaan high. Pada saat ditekan microcontroller akan menerima data input bernilai low dari switch tersebut, akan tetapi pada saat tidak ditekan nilainya undefined, cenderung float, antara low dengan high.

Resistor Pull-Down
Untuk kasus pertama, dimana nilai float terjadi pada kondisi low, perlu digunakan resistor pull-down. Resistor pull-down akan membuat nilai float menjadi nilai low. Dengan menambahkan sebuah resistor menuju ground, yang dirangkai paralel dengan jalur yang menuju input pin microcontroller.
Setelah menggunakan resistor pull-down, untuk kasus pertama, bila switch ditekan akan memberikan data input bernilai logika high pada microcontroller. Sementara pada saat tidak ditekan, nilai yang diterima microcontroller tidak lagi float, melainkan telah bernilai low. Cocok digunakan untuk aplikasi dengan kondisi default switch (saat tidak ditekan) bernilai low. Untuk external interrupt Arduino menggunakan switch dengan resistor pull-down, disarankan menggunakan mode operasi interrupt RISING. Dimana interrupt akan dijalankan saat switch ditekan data input berubah dari low menjadi high. Begitu switch tidak ditekan, akan bernilai low.

Resistor Pull-Up
Untuk kasus pertama, dimana nilai float terjadi pada kondisi high, perlu digunakan resistor pull-up. Resistor pull-up akan membuat nilai float menjadi nilai high. Dengan menambahkan sebuah resistor menuju sumber tegangan, yang dirangkai paralel dengan jalur yang menuju input pin microcontroller.
Setelah menggunakan resistor pull-up, untuk kasus kedua, bila switch ditekan akan memberikan data input bernilai logika low pada microcontroller. Sementara pada saat tidak ditekan, nilai yang diterima microcontroller tidak lagi float, melainkan telah bernilai high.

Cocok digunakan untuk aplikasi dengan kondisi default switch (saat tidak ditekan) bernilai high. Untuk external interrupt Arduino menggunakan switch dengan resistor pull-up, disarankan menggunakan mode operasi interrupt FALLING. Dimana interrupt akan dijalankan saat switch ditekan data input berubah dari high menjadi low. Begitu switch tidak ditekan, akan bernilai high.

Kesimpulan

Baik resistor pull-up maupun pull-down, keduanya sama-sama digunakan untuk mencegah terjadinya nilai float, undefined state, yang akan diolah untuk data input microcontroller. Perbedaan keduanya lebih pada aplikasi yang digunakan. Resistor pull-up digunakan untuk mencegah nilai float pada kondisi high dengan menambahkan sebuah resistor pada jalur sumber tegangan dan paralel dengan jalur input ke microcontroller. Sedangkan resistor pull-down digunakan untuk mencegah nilai float pada kondisi low dengan menambahkan sebuah resistor pada alur ke ground dan paralel dengan jalur input ke microcontroller.

2. Bouncing is the tendency of any two metal contacts in an electronic device to generate multiple signals as the contacts close or open; debouncing is any kind of hardware device or software that ensures that only a single signal will be acted upon for a single opening or closing of a contact. Adding a delay force the controller to stop for a particular time period, but adding delays is not a good option into the program, as it pause the program and increase the processing time. The best way is to use interrupts in the code for software bouncing.
3.

AND:

Performs a bitwise AND operation on the destination (first) and source (second) operands and stores the result in the destination operand location. The source operand can be an immediate, a register, or a memory location; the destination operand can be a register or a memory location.

(However, two memory operands cannot be used in one instruction.) Each bit of the result is set to 1 if both corresponding bits of the first and second operands are 1; otherwise, it is set to 0.

This instruction can be used with a LOCK prefix to allow the it to be executed atomically.

IN:

Copies the value from the I/O port specified with the second operand (source operand) to the destination operand (first operand). The source operand can be a byte-immediate or the DX register; the destination operand can be register AL, AX, or EAX, depending on the size of the port being accessed (8, 16, or 32 bits, respectively). Using the DX register as a source operand allows I/O port addresses from 0 to 65,535 to be accessed; using a byte immediate allows I/O port addresses 0 to 255 to be accessed.

When accessing an 8-bit I/O port, the opcode determines the port size; when accessing a 16- and 32-bit I/O port, the operand-size attribute determines the port size.

At the machine code level, I/O instructions are shorter when accessing 8-bit I/O ports. Here, the upper eight bits of the port address will be 0.

This instruction is only useful for accessing I/O ports located in the processor's I/O address space. See Chapter 13, Input/Output, in the IA-32 Intel Architecture Software Developer's Manual, Volume 1, for more information on accessing I/O ports in the I/O address space.

CALL:

Saves procedure linking information on the stack and branches to the procedure (called procedure) specified with the destination (target) operand. The target operand specifies the address of the first instruction in the called procedure. This operand can be an immediate value, a generalpurpose register, or a memory location.

This instruction can be used to execute four different types of calls:

Near call

A call to a procedure within the current code segment (the segment currently pointed to by the CS register), sometimes referred to as an intrasegment call.

Far call

A call to a procedure located in a different segment than the current code segment, sometimes referred to as an intersegment call.

Inter-privilege-level far call

A far call to a procedure in a segment at a different privilege level than that of the currently executing program or procedure.

Task switch

A call to a procedure located in a different task.

The latter two call types (inter-privilege-level call and task switch) can only be executed in protected mode. See the section titled "Calling Procedures Using Call and RET" in Chapter 6 of the IA-32 Intel Architecture Software Developer's Manual, Volume 1, for additional information on near, far, and inter-privilege-level calls. See Chapter 6, Task Management, in the IA-32 Intel Architecture Software Developer's Manual, Volume 3, for information on performing task switches with the CALL instruction.

CMP:

Compares the first source operand with the second source operand and sets the status flags in the EFLAGS register according to the results. The comparison is performed by subtracting the second operand from the first operand and then setting the status flags in the same manner as the SUB instruction. When an immediate value is used as an operand, it is sign-extended to the length of the first operand.

The CMP instruction is typically used in conjunction with a conditional jump (Jcc), condition move (CMOVcc), or SETcc instruction. The condition codes used by the Jcc, CMOVcc, and SETcc instructions are based on the results of a CMP instruction. Appendix B, EFLAGS Condition Codes, in the IA-32 Intel Architecture Software Developer's Manual, Volume 1, shows the relationship of the status flags and the condition codes.

JNZ:

conditional jump to operand if ZF==0

JZ:

conditional jump to operand if ZF==1