**Birla Institute Of Technology and Sciences, Pilani**

**CS F241-Microprocessor Programming and Interfacing**



P26-Elevator Control

**Submitted to**-

**Professor G Sai Sesha Chalapathi**

**Submitted by-**

**Rohit Lodha (2015A7PS040P)**

**Vikram Nitin(2015A7PS042P)**

**Aayushmaan Jain(2015A7PS043P)**

**Saurabh Raje(2015A7PS045P)**

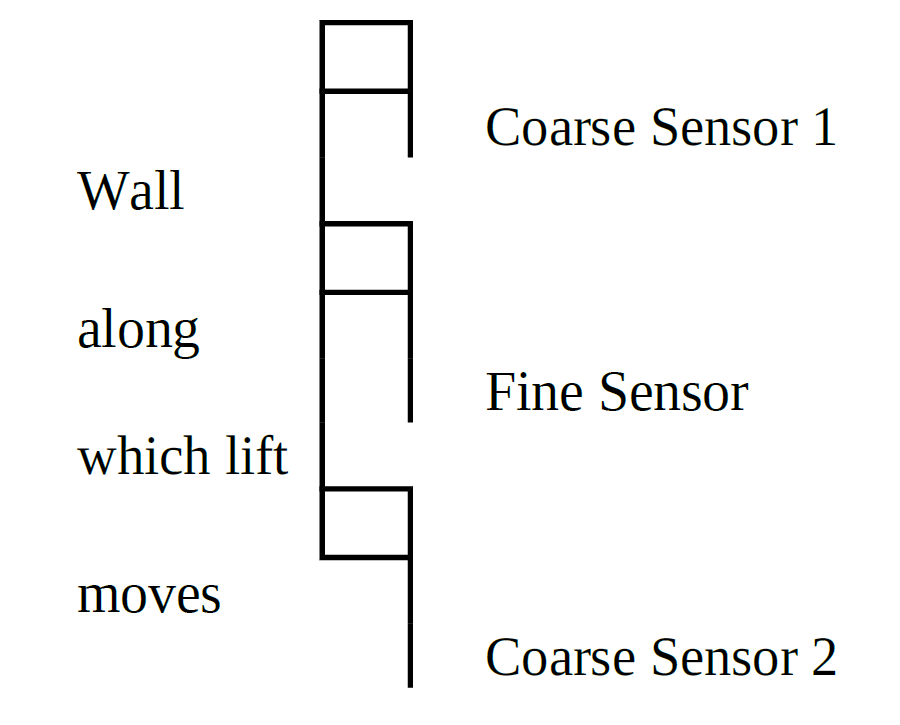
**ELEVATOR CONTROL**

**SYSTEM REQUIREMENTS**

* The elevator operates along **3 floors**.
* When not in use the elevator is always on the ground floor.
* The elevator can be called by pressing any one of **two buttons available on each floor**.
  + One button is **up** and the other is **down**.
* Whether the elevator stops at the floor or not depends on the direction in which the lift moves. For eg: if the lift is moving in upward direction and the person on say the 2nd floor presses the down button; the lift will not stop in the current journey. When the lift reaches the 3rd floor and starts moving down then the lift will stop at the 2nd floor.
* At **every floor there is a 7-segement display** that indicates the floor in which the lift is right now. The display can be any value from 0 - 3. ‘0’ indicates the ground floor.
* Inside the lift **- buttons are available for floor selection**.
* **The floor towards which the lift is moving is also displayed** within the lift.
* **Doors to the lift open and close automatically**.
* When the lift reaches any floor where it has to stop it opens automatically, and it closes when a button called “Door Close” is pressed. **Lift does not move until the door is closed.**
* System runs from a standard power inlet.

**SYSTEM SPECIFICATIONS**

* An Electro-magnetic system is used for open and close of the door. We have just provided the on/off control.
* A heavy duty servo motor is used for lift movement. We have just provided the input to the driver circuit.
* The inputs are direction (up/down) and a PWM input which control the speed at which the lift moves. The duty cycle can vary from 20% to 60%.
* The frequency of the PWM signal is 20 Hz.
* For detecting whether the lift has reached a floor, the system has a set of three sensors –
  + Two ‘coarse’ sensors and a ‘fine’ sensor. All the sensors are contact switches (i.e.) when the lift reaches the point where the sensors are placed, the contact switch gets pushed in. Output of contact switches are low when closed and high otherwise. The sensor arrangement is represented in the fig below

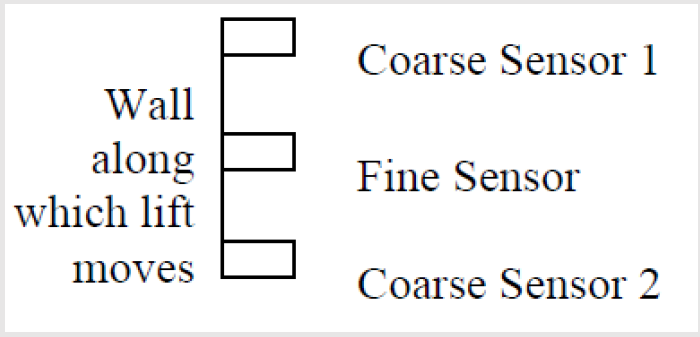


* On the ground floor – only Coarse Sensor1 and Fine Sensor will be available. On the 3rd floor only Coarse Sensor 2 and the Fine Sensor will be available.
* When the lift starts at the ground floor it starts at a low speed gradually accelerating to the maximum speed. It should operate at maximum speed when it reaches ‘Coarse Sensor 1”. As the lift moves up if it has to stop at floor ‘1’, when Coarse Sensor 2 is detected at that floor the lift starts moving at a low speed until it can stop when it reaches Fine sensor. When it starts again it moves at low speeds and reaches the maximum possible speed when it reaches the fine sensor. The same is done in the reverse direction with the appropriate sensors.
* Speed at which the lift moves is proportional to the duty cycle. For acceleration, duty cycle has to be gradually increased from 20 % to 60 %. And for deceleration, the duty cycle reduced from 60 % to 20 %. The increase is in steps of 20 %
* A 7447 chip (BCD to seven segment converter) is used for driving the 7-segment displays.
* 7447 takes a 4-bit BCD value and converts into the corresponding 7-segement equivalent.

**ASSUMPTIONS**

1. The Program starts at 0100H when the processor is reset.
2. Coarse and fine sensors that are available on each floor produce a binary output. Each sensor (coarse and fine) is assumed to be a push button, pushed by the elevator as it moves.

* Coarse and fine sensors have to be manually pressed during the simulation. (However, in reality they would be automatically pressed by the contact from the elevator, as it moves.)



1. Floor buttons – both in and outside the life – have been assumed to be push buttons, as well. The floor buttons are pressed by the user.

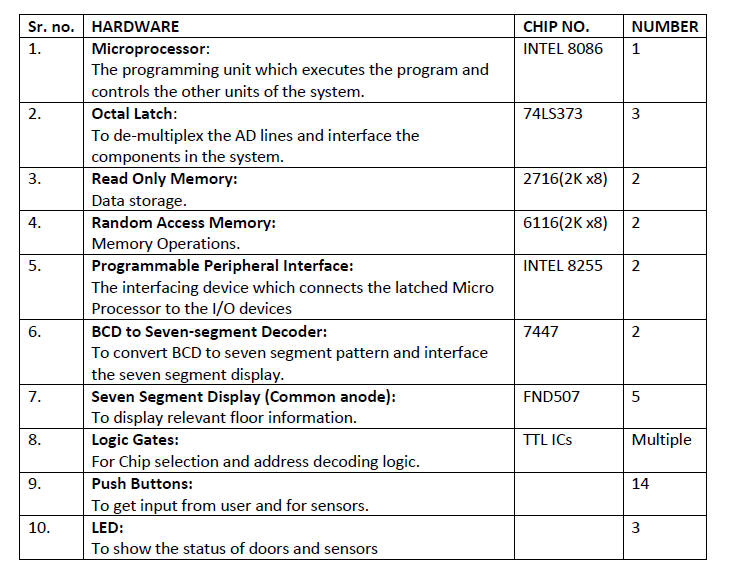
* While a button is not being pressed, it generates a logic 0 since it has been pulled down. This has been done to prevent the input from floating.

1. Unless the door is closed manually, the lift will not move.
2. Only one button is pressed at a given instant. However, any button on the floor may be pressed while the elevator is in motion.
3. By default the elevator is on the ground floor, and awaits user input.
4. Uni-directional motors have been used in this design project. Thus, to simulate both the upward and downward motion, two motors have been used – one for taking the lift up and one for taking it down.
5. PWM (Pulse Width Modulation) input is given through a pulse generator.
6. An Electro-magnetic system is used for opening and closing of the door. We have assumed that the doors open automatically, and have just provided a push button for closing the elevator door (just an on/off mechanism). We have also assumed that some mechanism exists that takes the door close button input and closes the door.
7. We have assumed that some mechanism involving a heavy duty servo motor already exists, and we have just provided an input to its driver circuit.
8. In the physical implementation, there will be separate sets of sensors for each floor. However, we have simplified the design to have just one set of sensors (one fine and two coarse). Since any two sets of sensors will never be triggered together, this is a reasonable simplification. These sensors can be used to accelerate or decelerate the lift, and the number of times the sensors are triggered can be used to infer the floor on which the lift is.

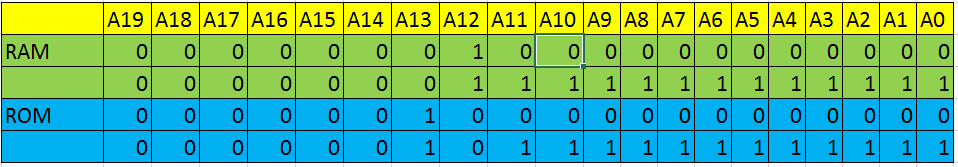
**Hardware Description**

* Two 8255 chips have been used – one having start address as 00H and the other as 10H.
  + The first 8255 chip has been used to interface the two motors – via a series of resistors. Two LEDS which indicate the status of the coarse sensors have also been connected to this chip. Its Port A and Port C are used for controlling the motors, and Port B has indicator LEDs for the contact sensors.
  + The second 8255 chip has been used to interface the 7-segment displays as well as the various (input) buttons and the coarse and fine sensors. Its Port A is used for controlling the seven segment displays, and Port C is used to access the sensor data. Port B is unused.
* 4KB of RAM has been used in this project – two 6116 chips. The RAM has been divided into two banks (odd and even) of 2KB each. The starting address of the RAM is 01000H.
* 4KB of ROM has been used in this project – two 2716 chips. The ROM has also been divided into two banks (odd and even) of 2KB each. The starting address of the ROM used is 02000H.
* 4 seven segment displays, one on each floor, indicate the floor the elevator is currently at.
  + One seven segment display inside the lift indicates the target floor (the floor to which the elevator is headed).
  + All these displays are interfaced using BCD to Seven Segment Display converters (7447).
* There are two sets of motors, one for the movement of the lift in upward direction, and one for the movement in the downward direction.
* There are ‘Up’ and ‘Down’ buttons outside the lift on each floor, except the third floor (which has no ‘Down’ button) and the ground floor (which has no ‘Up’ button’).
* In addition, there are four buttons (one for each floor) inside the lift to select the destination floor. Thus, there are 14 buttons in all.
* Two ‘coarse’ and one ‘fine’ sensor are used to track the movement of the elevator

**Components Used**



MEMORY MAPPING



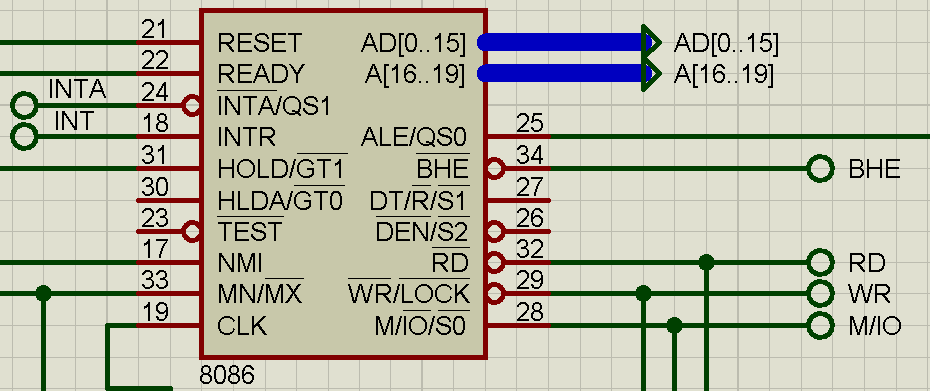
RAMeven  01000H, 01002H, 01004H, …………. 01FFEH

RAModd 01001H, 01003H, 01005H, …………. 01FFFH

ROMeven 02000H, 02002H, 02004H, …………. 02FFEH

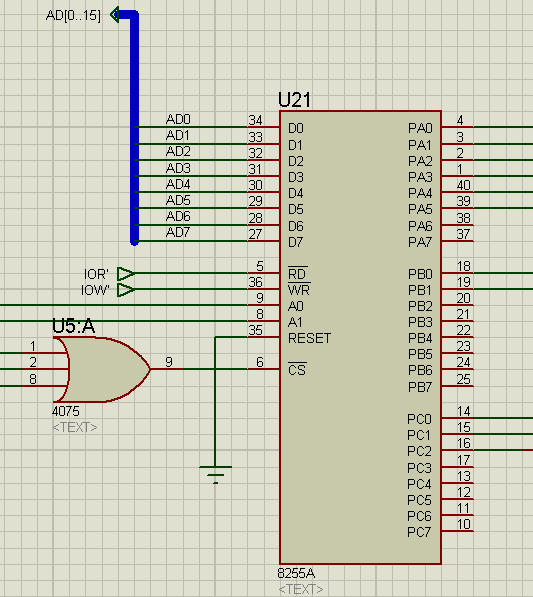
ROModd 02001H, 02003H, 02005H, …………. 02FFFH

**8086 – INTEL MICROPROCESSOR**



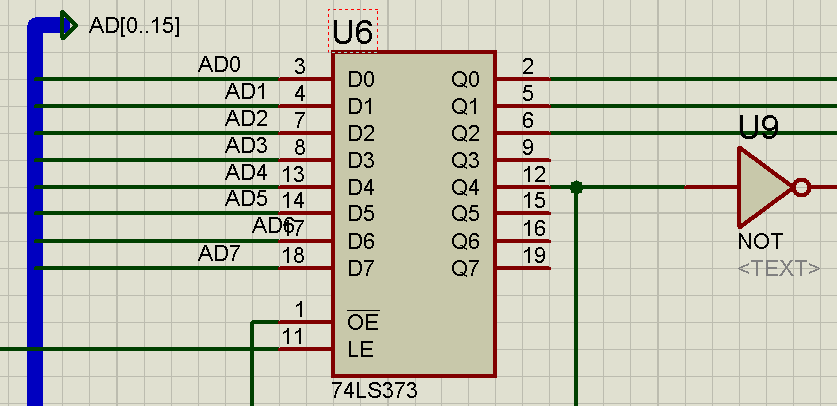
**Intel 8086 microprocessor** is a first member of x86 family of processors. The 8086 has complete 16-bit architecture - 16-bit internal registers, 16-bit data bus, and 20-bit address bus (1 MB of physical memory). Because the processor has 16-bit index registers and memory pointers, it can effectively address only 64 KB of memory.

**8255- INTEL Programmable Peripheral Interface chip**



The **Programmable Peripheral Interface (PPI) Intel 8255A** is a general purpose programmable I/O device which was designed to give the CPU access to programmable parallel I/O. It provides 24 I/O pins which may be individually programmed in 2 groups of 12 and used in 3 major modes of operation.

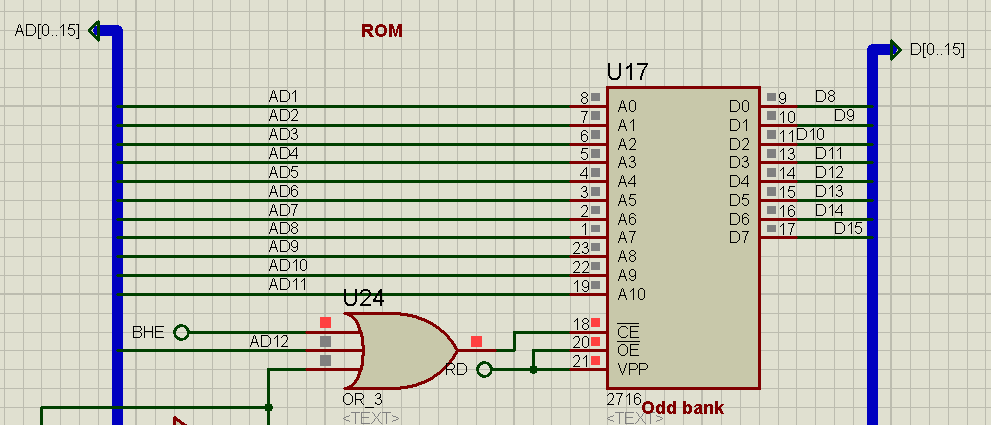
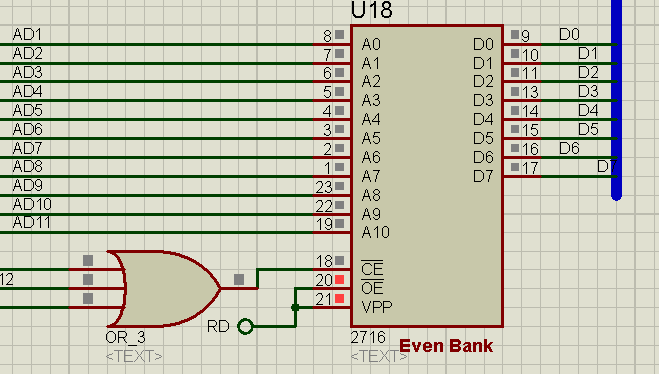
**74LS373- OCTAL D-TYPE LATCH**



The **74LS343 IC** consists of 8 latches with 3-state outputs for bus organized system applications. The flip-flops appear transparent to the data (data changes asynchronously) when Latch Enable (LE) is HIGH. When LE is LOW, the data that meets the setup times is latched. Data appears on the bus when the Output Enable (OE) is LOW. When OE is HIGH the bus output is in the high impedance state.

Two 74LS343 have been used in this design to de-multiplex the Address lines and interface the components in the system.

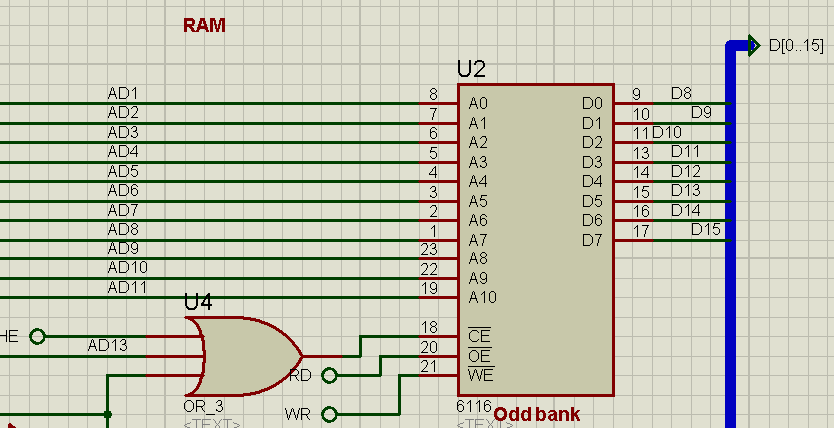
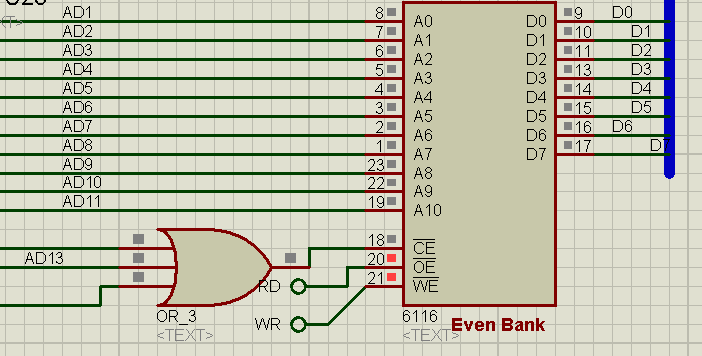
**2716-(ROM-READ ONLY MEMORY)**



The **2716** is a 16,384-bit EPROM organised as 2K x 8. A **read-only memory** is a type of non-volatile memory and is usually hard-wired. EPROM can be erased and re-programmed, but usually this can only be done at relatively slow speeds, may require special equipment to achieve, and is typically only possible a certain number of times.

Two such chips have been used in this project – one serves as the odd bank of memory and the other serves as the even bank of memory.

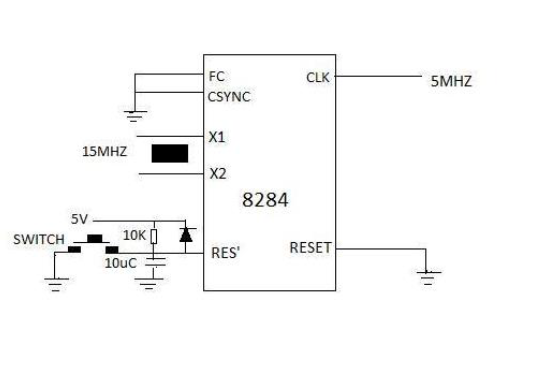
**6116-(RAM-RANDOM ACCESS MEMORY)**



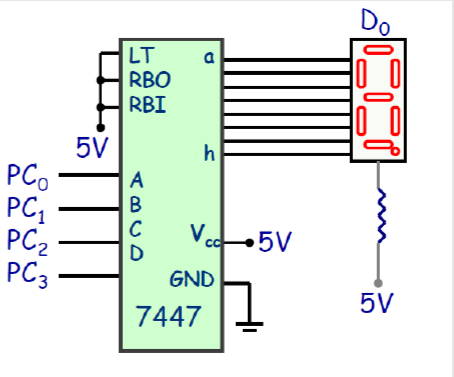
The 6116 is a 16,384-bit high speed static RAM organised as 2K x 8. A [**random-access**](https://en.wikipedia.org/wiki/Random_access)**memory device** allows [data](https://en.wikipedia.org/wiki/Data) items to be [read](https://en.wikipedia.org/wiki/Read_(computer)) or written in almost the same amount of time irrespective of the physical location of data inside the memory. RAM also allows for faster access of data.

Two such chips have been used in this project – one serves as the odd bank of memory and the other serves as the even bank of memory.

**8284 CLOCK GENERATOR**



**7-SEGMENT DISPLAY**



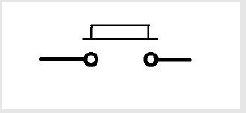
The 7447 IC take in a BCD value (4 bits) as input and outputs the corresponding seven-segment display value. Two 7447 ICs have been used.

4 – Seven segment displays have been used (one on each floor) to display the floor where the elevator has reached.

1 – Seven segment display has been set up inside the elevator which displays the destination floor where the elevator is headed

Two such chips have been used in this project – one for interfacing the motor and sensor LEDs, and the other for interfacing the various 7 segment displays and the push buttons.

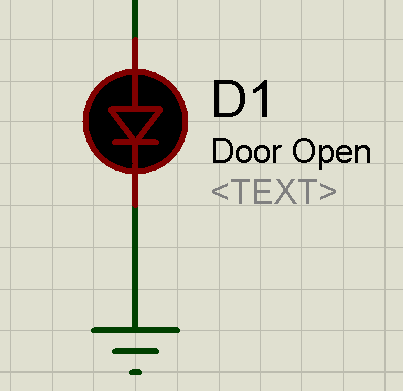
**PUSH BUTTON**



Each coarse and fine sensor, floor button outside and inside the elevator have been assumed to be a push button pushed by the elevator as it moves.

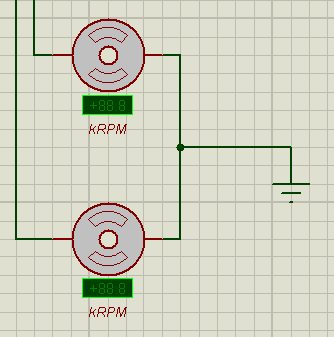
If not pressed, it generates a logic 0. This is being done to prevent the input from floating.

**LED-LIGHT EMITTING DIODE**



A **light-emitting diode (LED)** is a two-[lead](https://en.wikipedia.org/wiki/Lead_(electronics)) [semiconductor](https://en.wikipedia.org/wiki/Semiconductor) [light source](https://en.wikipedia.org/wiki/Light_source). It is a [p–n junction](https://en.wikipedia.org/wiki/P%E2%80%93n_junction) [diode](https://en.wikipedia.org/wiki/Diode) that emits light when activated.

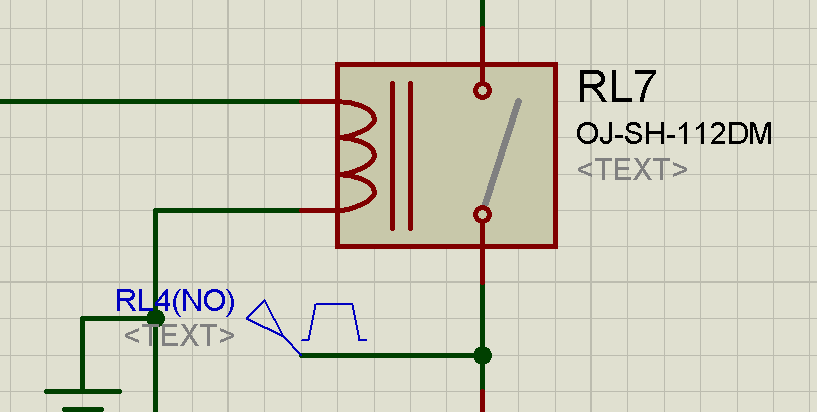
**HEAVY DUTY SERVO MOTOR**



A **servomotor** is a [rotary actuator](https://en.wikipedia.org/wiki/Rotary_actuator) or [linear actuator](https://en.wikipedia.org/wiki/Linear_actuator) that allows for precise control of angular or linear position, velocity and acceleration.  They are small in size but pack a big punch and are very energy-efficient.

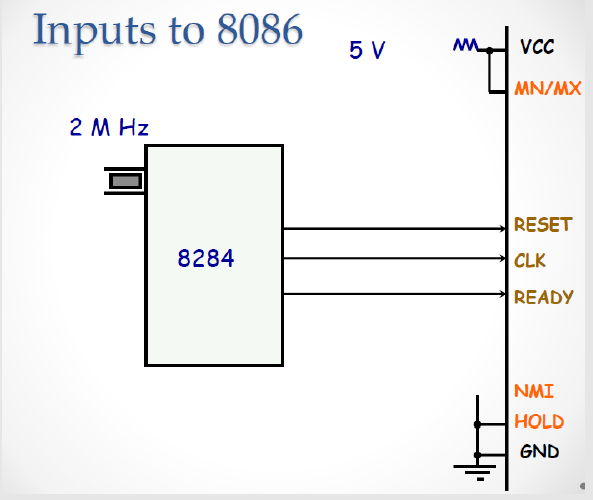
Two heavy duty servo motors have been used in this project – one for moving the elevator in the upward direction, and the other for moving it in the downward direction.

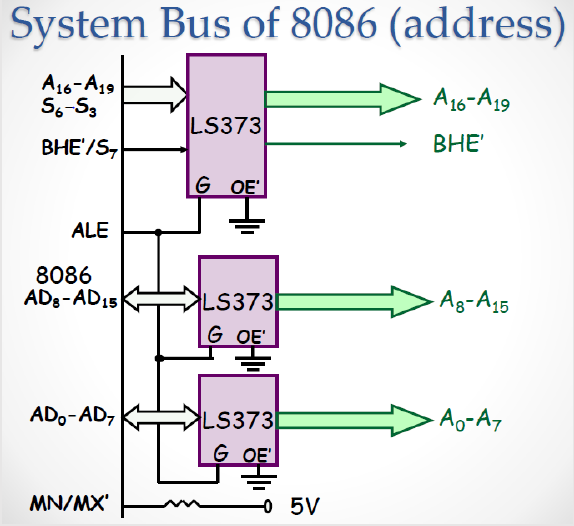
**RELAY**

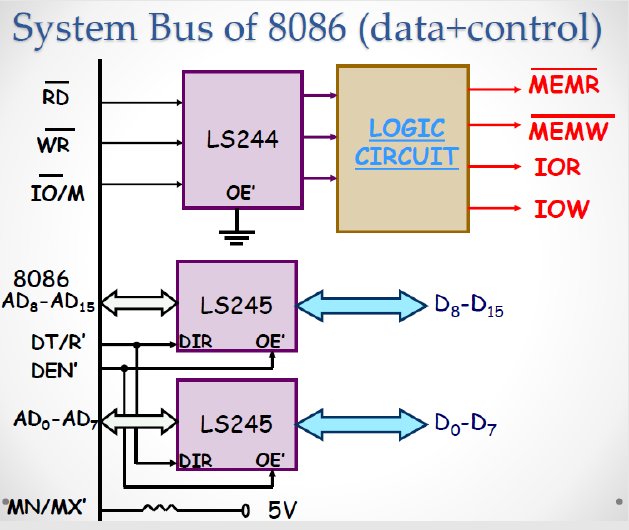


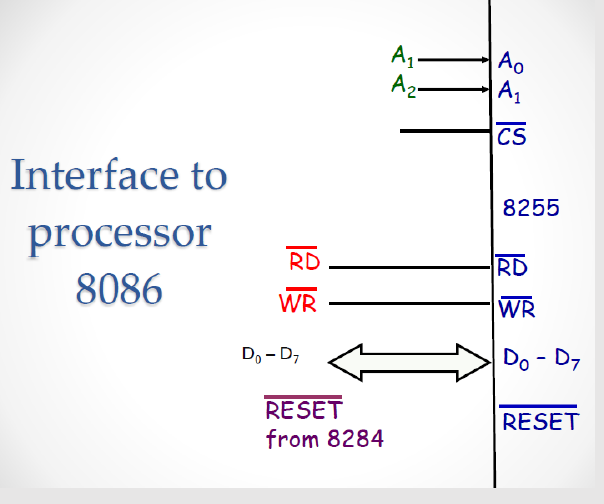
**Printed circuit board (PCB) relays** are compact relay devices used for power management in control system designs which require the relay to be mounted directly on the printed circuit board. In this projects the relays have been used to connect the two motors, and run them at different duty cycles.

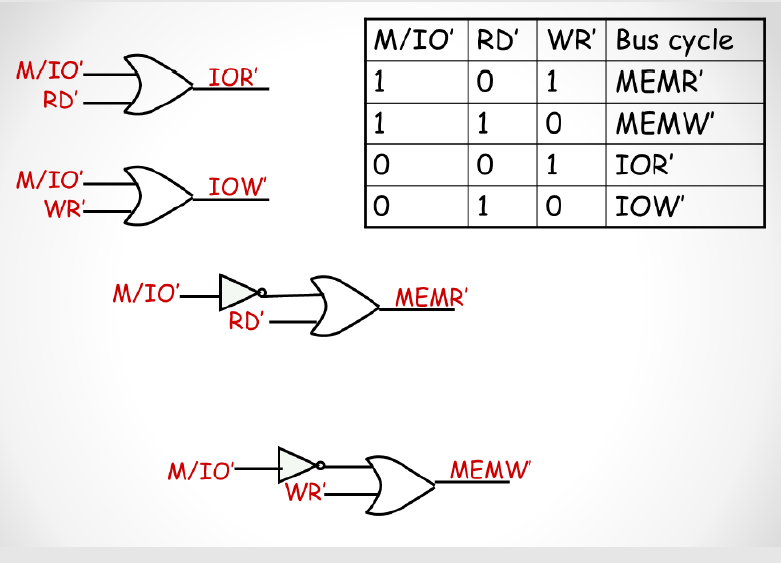
Different PWM signals – each with the same frequency but different duty cycles - have been used to power the two motors. Relays have been interfaced to different ports of the 8255 in order to actuate and control the duty cycle values. Depending upon the output of the ports, different relays get selected, and thus the duty cycle given to the motors change as well.

****

****







**CODE**

**.model tiny**

**.data**

**elevator\_position db 00h ;00h,01h,02h,03h**

**travel\_direction db 00h ;00(stop),01,10h**

**destination db 00h ;0fh(rest),00,01,02,03**

**new\_destination db ?**

**destination\_buffer db 8 dup (0fh)**

**buffer\_up db 8 dup (0fh)**

**buffer\_down db 8 dup (0fh)**

**destination\_size dw 0**

**up\_size dw 0**

**down\_size dw 0**

**accel\_stat db 00h**

**creg equ 06h**

**pa equ 00h**

**pb equ 02h**

**pc equ 04h**

**creg2 equ 16h**

**pa2 equ 10h**

**pb2 equ 12h**

**pc2 equ 14h**

**door\_close db 0bbh**

**up\_cat db 0e7h,0ebh,0edh**

**down\_cat db 0bdh,0beh,0eeh**

**inlift\_cat db 0d7h,0dbh,0ddh,0deh**

**sensor\_cat db 7eh,7dh,7bh**

**.code**

**.startup**

**;initialization**

**in0: mov al,88h**

**out creg,al**

**mov al,80h**

**out creg2,al**

**mov al,00h**

**out pa,al**

**mov al,00h**

**out pc2,al**

**;avoid key lockout**

**x0: mov al,00h**

**out pc,al**

**x1: in al, pc**

**and al,0f0h**

**cmp al,0f0h**

**jnz x1**

**;call delay\_20ms**

**mov dx,4544d**

**w1: nop**

**dec dx**

**jnz w1**

**;key press check**

**mov al,00h**

**out pc,al**

**x2: in al,pc**

**and al,0f0h**

**cmp al,0f0h**

**jz x2**

**;call delay\_20ms**

**mov dx,4544d**

**jf9: nop**

**dec dx**

**jnz jf9**

**mov al,00h**

**out pc,al**

**in al,pc**

**and al,0f0h**

**cmp al,0f0h**

**jz x2**

**;key press identification**

**;column 1**

**mov al,0eh**

**mov bl,al**

**out pc,al**

**in al,pc**

**and al,0f0h**

**cmp al,0f0h**

**jnz x3**

**;column 2**

**mov al,0dh**

**mov bl,al**

**out pc,al**

**in al,pc**

**and al,0f0h**

**cmp al,0f0h**

**jnz x3**

**;column 3**

**mov al,0bh**

**mov bl,al**

**out pc,al**

**in al,pc**

**and al,0f0h**

**cmp al,0f0h**

**jnz x3**

**;column 4**

**mov al,07h**

**mov bl,al**

**out pc,al**

**in al,pc**

**and al,0f0h**

**cmp al,0f0h**

**jz x2**

**;decode key**

**;generate hex code**

**x3: or al,bl**

**;up key pressed**

**mov cx,03h**

**mov di,00h**

**x4: cmp al,up\_cat[di]**

**jz x11**

**inc di**

**loop x4**

**;down key pressed**

**mov cx,03h**

**mov di,00h**

**x5: cmp al,down\_cat[di]**

**jz x12**

**inc di**

**loop x5**

**;inlift key pressed**

**mov cx,04h**

**mov di,00h**

**x6: cmp al,inlift\_cat[di]**

**jz x13**

**inc di**

**loop x6**

**;sensor key pressed**

**mov cx,03h**

**mov di,00h**

**x7: cmp al,sensor\_cat[di]**

**jz x14**

**inc di**

**loop x7**

**jmp x15**

**;operate according to the key type pressed**

**;up key press operations**

**x11: mov dl,travel\_direction**

**cmp dl,00h**

**jnz up6**

**;if not moving**

**mov ax,di**

**cmp al,elevator\_position**

**jz up4**

**mov destination,al**

**mov cl,4**

**rol al,cl**

**mov bl,elevator\_position**

**or al,bl**

**out pa,al**

**;set traveling direction**

**mov al,destination**

**cmp al,elevator\_position**

**jle up3**

**mov travel\_direction,01h ;move elv up**

**mov al,01h**

**out pc2,al**

**jmp up5**

**up3: mov travel\_direction,10h ;move elv down**

**mov al,02h**

**out pc2,al**

**jmp up5**

**up4: mov al,04h ;elv at pos, just open door**

**out pc2,al**

**jmp up5**

**;if moving**

**up6: mov ax,di**

**mov di,00**

**cmp up\_size,00h**

**jz igr1**

**mov cx,up\_size**

**up1: cmp al,buffer\_up[di]**

**jz up5**

**inc di**

**loop up1**

**igr1: mov buffer\_up[di],al**

**inc up\_size**

**up5: jmp x16**

**;down key press operations**

**;if not moving**

**x12: inc di**

**mov dl,travel\_direction**

**cmp dl,00h**

**jnz dwn6**

**;if not moving**

**mov ax,di**

**cmp al,elevator\_position**

**jz dwn4**

**mov destination,al**

**mov cl,4**

**rol al,cl**

**mov bl,elevator\_position**

**or al,bl**

**out pa,al**

**;set traveling direction**

**mov al,destination**

**cmp al,elevator\_position**

**jle dwn3**

**mov travel\_direction,01h ;move elv up**

**mov al,01h**

**out pc2,al**

**jmp dwn5**

**dwn3: mov travel\_direction,10h ;move elv down**

**mov al,02h**

**out pc2,al**

**jmp dwn5**

**dwn4: mov al,04h ;elv at pos, just open door**

**out pc2,al**

**jmp dwn5**

**;if moving**

**dwn6: mov ax,di**

**mov di,00**

**cmp down\_size,00h**

**jz igr2**

**mov cx,down\_size**

**dwn1: cmp al,buffer\_down[di]**

**jz dwn5**

**inc di**

**loop dwn1**

**igr2: mov buffer\_down[di],al**

**inc down\_size**

**dwn5: jmp x16**

**;inlift key press operations**

**x13: cmp travel\_direction,00h**

**jz inl1**

**;if moving, check if floor requested is in queue**

**;ckeck in buffer\_up**

**mov ax,di**

**mov di,00h**

**cmp up\_size,00h**

**jz inl7**

**mov cx,up\_size**

**inl5: cmp al,buffer\_up[di]**

**jz inl2**

**inc di**

**loop inl5**

**inl7:**

**;check in buffer\_down**

**mov di,00h**

**cmp down\_size,00h**

**jz inl8**

**mov cx,down\_size**

**inl6: cmp al,buffer\_down[di]**

**jz inl2**

**inc di**

**loop inl6**

**inl8:**

**;check destination**

**cmp al,destination**

**jz inl2**

**;check in destination\_buffer**

**mov ax,di**

**mov di,00**

**cmp destination\_size,00h**

**jz igr3**

**mov cx,destination\_size**

**inl4: cmp al,destination\_buffer[di]**

**jz inl2**

**inc di**

**loop inl4**

**igr3: mov destination\_buffer[di],al**

**inc destination\_size**

**jmp inl2**

**inl1: mov ax,di**

**cmp al,elevator\_position**

**jz inl2**

**cmp al,elevator\_position**

**jg inl3**

**mov travel\_direction,10h**

**mov destination,al**

**jmp inl2**

**inl3: mov travel\_direction,01h**

**mov destination,al**

**inl2: jmp x16**

**;sensor key press operations**

**x14: cmp di,00h**

**jnz co1**

**;coarse sensor 1 pressed**

**;check travelling direction**

**cmp travel\_direction,01h**

**jnz goDown**

**;if travelling up**

**;light coarse 1 led**

**mov al,01h**

**out pb2,al**

**;check acceleration status**

**cmp accel\_stat,00h**

**jnz co3**

**;if not accelerating**

**;close 20% generator**

**mov al,00h**

**out pc2,al**

**;start 30% generator**

**;mov al,01h**

**;out pa2,al**

**;wait**

**mov dx,56800d**

**jf8: nop**

**dec dx**

**jnz jf8**

**;start 40% generator**

**mov al,04h**

**out pa2,al**

**;wait**

**mov dx,56800d**

**jf7: nop**

**dec dx**

**jnz jf7**

**;stop 40% generator**

**mov al,00h**

**out pa2,al**

**;start 60% generator**

**mov al,10h**

**out pa2,al**

**mov accel\_stat,01h**

**;if travelling down**

**;see if needs deceleration**

**goDown: mov al,elevator\_position**

**dec al**

**;check with destination**

**cmp al,destination**

**jnz co4**

**;deceleration required**

**co6: ;light coarse 1 led**

**mov al,01h**

**out pb2,al**

**;start 40% generator**

**mov al,08h**

**out pa2,al**

**;wait**

**mov dx,56800d**

**jf6: nop**

**dec dx**

**jnz jf6**

**;start 30% generator**

**;mov al,02h**

**;out pa2,al**

**;wait**

**mov dx,56800d**

**jf5: nop**

**dec dx**

**jnz jf5**

**;stop 40% generator**

**mov al,00h**

**out pa2,al**

**;stop 30% generator**

**;mov al,00h**

**;out pa2,al**

**;start 20% generator**

**mov al,02h**

**out pc2,al**

**mov accel\_stat,00h**

**jmp co3**

**;check in destination\_buffer**

**co4: mov di,00h**

**cmp destination\_size,00h**

**jz co5**

**mov cx,destination\_size**

**co7: cmp al,destination\_buffer[di]**

**jz co6 ;if found go for deceleration**

**inc di**

**loop co7**

**;check in buffer\_down**

**co5: mov di,00h**

**cmp down\_size,00h**

**jz co3**

**mov cx,down\_size**

**co8: cmp al,buffer\_down[di]**

**jz co6 ;if found go for deceleration**

**inc di**

**loop co8**

**co3: jmp x16**

**co1: cmp di,01h**

**jnz co2**

**;fine sensor pressed**

**mov al,travel\_direction**

**cmp al,01h**

**jnz sen1**

**;if moving up**

**;update display**

**inc elevator\_position**

**mov al,elevator\_position**

**mov bl,destination**

**mov cl,4**

**rol bl,cl**

**or al,bl**

**out pa,al**

**;see if it has to stop**

**mov al,elevator\_position**

**cmp al,destination**

**jnz alt1**

**;stop**

**mov al,04h**

**out pc2,al**

**;update destination**

**mov al,elevator\_position**

**mov new\_destination,al**

**;look in destination\_buffer**

**mov dl,new\_destination**

**mov di,00h**

**cmp destination\_size,00h**

**jz igr4**

**mov cx,destination\_size**

**p2: cmp dl,destination\_buffer[di]**

**jg p1**

**mov dl,destination\_buffer[di]**

**mov si,di**

**p1: inc di**

**loop p2**

**igr4: mov new\_destination,dl**

**mov al,elevator\_position**

**cmp new\_destination,al**

**jz sen6 ;not found in destination\_buffer**

**;found in destination\_buffer**

**;remove from destination\_buffer**

**mov di,si**

**mov cx,destination\_size**

**dec cx**

**cmp cx,00h**

**jz ign5**

**p3: mov al,destination\_buffer[di+1]**

**mov destination\_buffer[di],al**

**inc di**

**cmp di,cx**

**jnz p3**

**ign5: mov destination\_buffer[di],0fh**

**dec destination\_size**

**mov al,new\_destination**

**mov destination,al**

**jmp sen2**

**;not found in destination\_buffer**

**;look in buffer\_up**

**sen6: mov dl,new\_destination**

**mov di,00h**

**cmp up\_size,00h**

**jz igr5**

**mov cx,up\_size**

**p5: cmp dl,buffer\_up[di]**

**jg p4**

**mov dl,buffer\_up[di]**

**mov si,di**

**p4: inc di**

**loop p5**

**igr5: mov new\_destination,dl**

**mov al,elevator\_position**

**cmp new\_destination,al**

**jz sen7 ;not found in buffer\_up**

**;found in buffer\_up**

**;remove from buffer\_up**

**mov di,si**

**mov cx,up\_size**

**dec cx**

**cmp cx,00h**

**jz ign6**

**p6: mov al,buffer\_up[di+1]**

**mov buffer\_up[di],al**

**inc di**

**cmp di,cx**

**jnz p6**

**ign6: mov buffer\_up[di],0fh**

**dec up\_size**

**mov al,new\_destination**

**mov destination,al**

**jmp sen2**

**;not found in buffer\_up**

**;look in buffer\_down**

**sen7: mov dl,new\_destination**

**mov di,00h**

**cmp down\_size,00h**

**jz igr6**

**mov cx,down\_size**

**p8: cmp dl,buffer\_down[di]**

**jg p7**

**mov dl,buffer\_down[di]**

**mov si,di**

**p7: inc di**

**loop p8**

**igr6: mov new\_destination,dl**

**mov al,elevator\_position**

**cmp new\_destination,al**

**jz sen8 ;not found in buffer\_down**

**;found in buffer\_down**

**;remove from buffer\_down**

**mov di,si**

**mov cx,down\_size**

**dec cx**

**cmp cx,00h**

**jz ign7**

**p9: mov al,buffer\_down[di+1]**

**mov buffer\_down[di],al**

**inc di**

**cmp di,cx**

**jnz p9**

**ign7: mov buffer\_down[di],0fh**

**dec down\_size**

**mov al,new\_destination**

**mov destination,al**

**jmp sen2**

**;no place up to go**

**sen8: mov travel\_direction,10h ;go down now**

**;look in destination\_buffer**

**mov dl,new\_destination**

**mov di,00h**

**cmp destination\_size,00h**

**jz igr7**

**mov cx,destination\_size**

**q2: cmp dl,destination\_buffer[di]**

**jl q1**

**mov dl,destination\_buffer[di]**

**mov si,di**

**q1: inc di**

**loop q2**

**igr7: mov new\_destination,dl**

**mov al,elevator\_position**

**cmp new\_destination,al**

**jz ben6 ;not found in destination\_buffer**

**;found in destination\_buffer**

**;remove from destination\_buffer**

**mov di,si**

**mov cx,destination\_size**

**dec cx**

**cmp cx,00h**

**jz ign8**

**q3: mov al,destination\_buffer[di+1]**

**mov destination\_buffer[di],al**

**inc di**

**cmp di,cx**

**jnz q3**

**ign8: mov destination\_buffer[di],0fh**

**dec destination\_size**

**mov al,new\_destination**

**mov destination,al**

**jmp sen2**

**;not found in destination\_buffer**

**;look in buffer\_up**

**ben6: mov dl,new\_destination**

**mov di,00h**

**cmp up\_size,00h**

**jz igr8**

**mov cx,up\_size**

**q5: cmp dl,buffer\_up[di]**

**jl q4**

**mov dl,buffer\_up[di]**

**mov si,di**

**q4: inc di**

**loop q5**

**igr8: mov new\_destination,dl**

**mov al,elevator\_position**

**cmp new\_destination,al**

**jz ben7 ;not found in buffer\_up**

**;found in buffer\_up**

**;remove from buffer\_up**

**mov di,si**

**mov cx,up\_size**

**dec cx**

**cmp cx,00h**

**jz ign9**

**q6: mov al,buffer\_up[di+1]**

**mov buffer\_up[di],al**

**inc di**

**cmp di,cx**

**jnz q6**

**ign9: mov buffer\_up[di],0fh**

**dec up\_size**

**mov al,new\_destination**

**mov destination,al**

**jmp sen2**

**;not found in buffer\_up**

**;look in buffer\_down**

**ben7: mov dl,new\_destination**

**mov di,00h**

**cmp down\_size,00h**

**jz igr9**

**mov cx,down\_size**

**q8: cmp dl,buffer\_down[di]**

**jl q7**

**mov dl,buffer\_down[di]**

**mov si,di**

**q7: inc di**

**loop q8**

**igr9: mov new\_destination,dl**

**mov al,elevator\_position**

**cmp al,new\_destination**

**jz ben8 ;not found in buffer\_down**

**;found in buffer\_down**

**;remove from buffer\_down**

**mov di,si**

**mov cx,down\_size**

**dec cx**

**cmp cx,00h**

**jz ign10**

**q9: mov al,buffer\_down[di+1]**

**mov buffer\_down[di],al**

**inc di**

**cmp di,cx**

**jnz q9**

**ign10: mov buffer\_down[di],0fh**

**dec down\_size**

**mov al,new\_destination**

**mov destination,al**

**jmp sen2**

**;nowhere to go, rest**

**ben8: mov al,new\_destination**

**mov destination,al**

**mov travel\_direction,00h**

**jmp sen2**

**;look in buffs**

**;look in destination\_buffer**

**alt1: mov al,elevator\_position**

**mov di,00h**

**cmp destination\_size,00h**

**jz igr10**

**mov cx,destination\_size**

**alt3: cmp al,destination\_buffer[di]**

**jz alt2**

**inc di**

**loop alt3**

**igr10: jmp alt5**

**;found it? remove**

**alt2: mov cx,destination\_size**

**dec cx**

**cmp cx,00h**

**jz ign1**

**alt4: mov al,destination\_buffer[di+1]**

**mov destination\_buffer[di],al**

**inc di**

**cmp di,cx**

**jnz alt4**

**ign1: mov destination\_buffer[di],0fh**

**dec destination\_size**

**mov al,04h**

**out pc2,al**

**jmp sen2**

**;look in buffer\_up**

**alt5: mov al,elevator\_position**

**mov di,00h**

**cmp up\_size,00h**

**jz igr11**

**mov cx,up\_size**

**alt7: cmp al,buffer\_up[di]**

**jz alt6**

**inc di**

**loop alt7**

**igr11: jmp sen2**

**;found it? remove**

**alt6: mov cx,up\_size**

**dec cx**

**cmp cx,00h**

**jz ign2**

**alt8: mov al,buffer\_up[di+1]**

**mov buffer\_up[di],al**

**inc di**

**cmp di,cx**

**jnz alt8**

**ign2: mov buffer\_up[di],0fh**

**dec up\_size**

**mov al,04h**

**out pc2,al**

**jmp sen2**

**sen1:**

**;if moving down**

**;update display**

**dec elevator\_position**

**mov al,elevator\_position**

**mov bl,destination**

**mov cl,4**

**rol bl,cl**

**or al,bl**

**out pa,al**

**;see if it has to stop**

**mov al,elevator\_position**

**cmp al,destination**

**jnz act1**

**;stop**

**mov al,04h**

**out pc2,al**

**;update destination**

**mov al,elevator\_position**

**mov new\_destination,al**

**;look in destination\_buffer**

**mov dl,new\_destination**

**mov di,00h**

**cmp destination\_size,00h**

**jz igr13**

**mov cx,destination\_size**

**m2: cmp dl,destination\_buffer[di]**

**jl m1**

**mov dl,destination\_buffer[di]**

**mov si,di**

**m1: inc di**

**loop m2**

**igr13: mov new\_destination,dl**

**mov al,elevator\_position**

**cmp new\_destination,al**

**jz sen3 ;not found in destination\_buffer**

**;found in destination\_buffer**

**;remove from destination\_buffer**

**mov di,si**

**mov cx,destination\_size**

**dec cx**

**cmp cx,00h**

**jz ign11**

**m3: mov al,destination\_buffer[di+1]**

**mov destination\_buffer[di],al**

**inc di**

**cmp di,cx**

**jnz m3**

**ign11: mov destination\_buffer[di],0fh**

**dec destination\_size**

**mov al,new\_destination**

**mov destination,al**

**jmp sen2**

**;not found in destination\_buffer**

**;look in buffer\_up**

**sen3: mov dl,new\_destination**

**mov di,00h**

**cmp up\_size,00h**

**jz igr14**

**mov cx,up\_size**

**m5: cmp dl,buffer\_up[di]**

**jl m4**

**mov dl,buffer\_up[di]**

**mov si,di**

**m4: inc di**

**loop m5**

**igr14: mov new\_destination,dl**

**mov al,elevator\_position**

**cmp new\_destination,al**

**jz sen4 ;not found in buffer\_up**

**;found in buffer\_up**

**;remove from buffer\_up**

**mov di,si**

**mov cx,up\_size**

**dec cx**

**cmp cx,00h**

**jz ign12**

**m6: mov al,buffer\_up[di+1]**

**mov buffer\_up[di],al**

**inc di**

**cmp di,cx**

**jnz m6**

**ign12: mov buffer\_up[di],0fh**

**dec up\_size**

**mov al,new\_destination**

**mov destination,al**

**jmp sen2**

**;not found in buffer\_up**

**;look in buffer\_down**

**sen4: mov dl,new\_destination**

**mov di,00h**

**cmp down\_size,00h**

**jz igr15**

**mov cx,down\_size**

**m8: cmp dl,buffer\_down[di]**

**jl m7**

**mov dl,buffer\_down[di]**

**mov si,di**

**m7: inc di**

**loop m8**

**igr15: mov new\_destination,dl**

**mov al,elevator\_position**

**cmp new\_destination,al**

**jz sen5 ;not found in buffer\_down**

**;found in buffer\_down**

**;remove from buffer\_down**

**mov di,si**

**mov cx,down\_size**

**dec cx**

**cmp cx,00h**

**jz ign13**

**m9: mov al,buffer\_down[di+1]**

**mov buffer\_down[di],al**

**inc di**

**cmp di,cx**

**jnz m9**

**ign13: mov buffer\_down[di],0fh**

**dec down\_size**

**mov al,new\_destination**

**mov destination,al**

**jmp sen2**

**;no place up to go**

**sen5: mov travel\_direction,10h ;go down now**

**;look in destination\_buffer**

**mov dl,new\_destination**

**mov di,00h**

**cmp destination\_size,00h**

**jz igr16**

**mov cx,destination\_size**

**n2: cmp dl,destination\_buffer[di]**

**jg n1**

**mov dl,destination\_buffer[di]**

**mov si,di**

**n1: inc di**

**loop n2**

**igr16: mov new\_destination,dl**

**mov al,elevator\_position**

**cmp new\_destination,al**

**jz ben3 ;not found in destination\_buffer**

**;found in destination\_buffer**

**;remove from destination\_buffer**

**mov di,si**

**mov cx,destination\_size**

**dec cx**

**cmp cx,00h**

**jz ign14**

**n3: mov al,destination\_buffer[di+1]**

**mov destination\_buffer[di],al**

**inc di**

**cmp di,cx**

**jnz n3**

**ign14: mov destination\_buffer[di],0fh**

**dec destination\_size**

**mov al,new\_destination**

**mov destination,al**

**jmp sen2**

**;not found in destination\_buffer**

**;look in buffer\_up**

**ben3: mov dl,new\_destination**

**mov di,00h**

**cmp up\_size,00h**

**jz igr17**

**mov cx,up\_size**

**n5: cmp dl,buffer\_up[di]**

**jg n4**

**mov dl,buffer\_up[di]**

**mov si,di**

**n4: inc di**

**loop n5**

**igr17: mov new\_destination,dl**

**mov al,elevator\_position**

**cmp new\_destination,al**

**jz ben4 ;not found in buffer\_up**

**;found in buffer\_up**

**;remove from buffer\_up**

**mov di,si**

**mov cx,up\_size**

**dec cx**

**cmp cx,00h**

**jz ign15**

**n6: mov al,buffer\_up[di+1]**

**mov buffer\_up[di],al**

**inc di**

**cmp di,cx**

**jnz n6**

**ign15: mov buffer\_up[di],0fh**

**dec up\_size**

**mov al,new\_destination**

**mov destination,al**

**jmp sen2**

**;not found in buffer\_up**

**;look in buffer\_down**

**ben4: mov dl,new\_destination**

**mov di,00h**

**cmp down\_size,00h**

**jz igr18**

**mov cx,down\_size**

**n8: cmp dl,buffer\_down[di]**

**jg n7**

**mov dl,buffer\_down[di]**

**mov si,di**

**n7: inc di**

**loop n8**

**igr18: mov new\_destination,dl**

**mov al,elevator\_position**

**cmp new\_destination,al**

**jz ben5 ;not found in buffer\_down**

**;found in buffer\_down**

**;remove from buffer\_down**

**mov di,si**

**mov cx,down\_size**

**dec cx**

**cmp cx,00h**

**jz ign16**

**n9: mov al,buffer\_down[di+1]**

**mov buffer\_down[di],al**

**inc di**

**cmp di,cx**

**jnz n9**

**ign16: mov buffer\_down[di],0fh**

**dec down\_size**

**mov al,new\_destination**

**mov destination,al**

**jmp sen2**

**;nowhere to go, rest**

**ben5: mov al,new\_destination**

**mov destination,al**

**mov travel\_direction,00h**

**jmp sen2**

**;look in buffs**

**;look in destination\_buffer**

**act1: mov al,elevator\_position**

**mov di,00h**

**cmp destination\_size,00h**

**jz igr19**

**mov cx,destination\_size**

**act3: cmp al,destination\_buffer[di]**

**jz act2**

**inc di**

**loop act3**

**igr19: jmp act5**

**;found it? remove**

**act2: mov cx,destination\_size**

**dec cx**

**cmp cx,00h**

**jz ign3**

**act4: mov al,destination\_buffer[di+1]**

**mov destination\_buffer[di],al**

**inc di**

**cmp di,cx**

**jnz act4**

**ign3: mov destination\_buffer[di],0fh**

**dec destination\_size**

**mov al,04h**

**out pc2,al**

**jmp sen2**

**;look in buffer\_down**

**act5: mov al,elevator\_position**

**mov di,00h**

**cmp down\_size,00h**

**jz igr21**

**mov cx,down\_size**

**act8: cmp al,buffer\_down[di]**

**jz act6**

**inc di**

**loop act8**

**igr21: jmp sen2**

**;found it? remove**

**act6: mov cx,down\_size**

**dec cx**

**cmp cx,00h**

**jz ign4**

**act7: mov al,buffer\_down[di+1]**

**mov buffer\_down[di],al**

**inc di**

**cmp di,cx**

**jnz act7**

**ign4: mov buffer\_down[di],0fh**

**dec down\_size**

**mov al,04h**

**out pc2,al**

**sen2: jmp x16**

**;coarse sensor 2 pressed**

**;check for travelling direction**

**co2: cmp travel\_direction,10h**

**jnz goUp**

**;if travelling down**

**;light coarse 2 led**

**mov al,02h**

**out pb2,al**

**;check acceleration status**

**cmp accel\_stat,00h**

**jnz cos3**

**;if not accelerating**

**;stop 20% generator**

**mov al,00h**

**out pc2,al**

**;start 30% generator**

**;mov al,02h**

**;out pa2,al**

**;wait**

**mov dx,56800d**

**jf1: nop**

**dec dx**

**jnz jf1**

**;start 40% generator**

**mov al,08h**

**out pa2,al**

**;wait**

**mov dx,56800d**

**jf2: nop**

**dec dx**

**jnz jf2**

**;stop 40% generator**

**mov al,00h**

**out pa2,al**

**;start 60% generator**

**mov al,20h**

**out pa2,al**

**;update acceleration status**

**mov accel\_stat,01h**

**jmp cos3**

**;if travelling up**

**goUp: mov al,elevator\_position**

**inc al**

**;check with destination**

**cmp al,destination**

**jnz cos4**

**;deceleration required**

**cos6: ;light coarse 2 led**

**mov al,02h**

**out pb2,al**

**;start 40% generator**

**mov al,04h**

**out pa2,al**

**;wait**

**mov dx,56800d**

**jf3: nop**

**dec dx**

**jnz jf3**

**;start 30% generator**

**;mov al,01h**

**;out pa2,al**

**;wait**

**mov dx,56800d**

**jf4: nop**

**dec dx**

**jnz jf4**

**;stop 30% generator**

**;mov al,00h**

**;out pa2,al**

**;stop 40% generator**

**mov al,00h**

**out pa2,al**

**;start 20% generator**

**mov al,01h**

**out pc2,al**

**mov accel\_stat,00h**

**jmp cos3**

**;check in destination\_buffer**

**cos4: mov di,00h**

**cmp destination\_size,00h**

**jz cos5**

**mov cx,destination\_size**

**cos7: cmp al,destination\_buffer[di]**

**jz cos6 ;if found go for deceleration**

**inc di**

**loop cos7**

**;check in buffer\_up**

**cos5: mov di,00h**

**cmp up\_size,00h**

**jz cos3**

**mov cx,up\_size**

**cos8: cmp al,buffer\_up[di]**

**jz cos6 ;if found go for deceleration**

**inc di**

**loop cos8**

**cos3: jmp x16**

**;door close key press operations**

**x15:**

**cmp travel\_direction,00h**

**jz dor1**

**cmp travel\_direction,01h**

**jz dor3**

**mov al,destination**

**mov cl,4**

**rol al,cl**

**mov bl,elevator\_position**

**or al,bl**

**out pa,al**

**mov al,02h**

**out pc2,al**

**jmp x16**

**dor3: mov al,destination**

**mov cl,4**

**rol al,cl**

**mov bl,elevator\_position**

**or al,bl**

**out pa,al**

**mov al,01h**

**out pc2,al**

**jmp x16**

**dor1: mov destination,00h**

**mov elevator\_position,00h**

**mov travel\_direction,00h**

**jmp in0**

**;mov al,elevator\_position**

**;cmp al,destination**

**;jz dor2**

**;mov al,02h**

**;out pc2,al**

**;mov al,00h**

**;mov cl,4**

**;rol al,cl**

**;mov bl,elevator\_position**

**;or al,bl**

**;out pa,al**

**;jmp x16**

**dor2: mov al,04h**

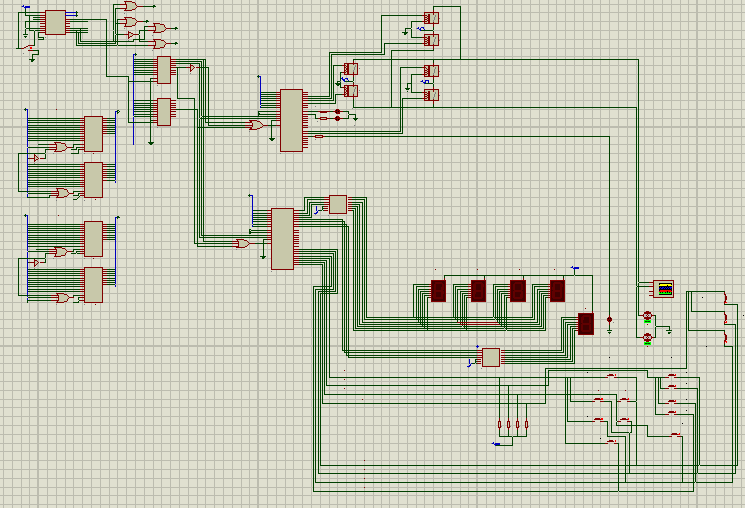
**out pc2,al**

**x16: jmp x0**

**.exit**

**End**

**DESIGN**

****