**(10) Question 1.**

**(2) Part a)** V = I\*R

**(2) Part b)** P = V\*I

**(2) Part c)** -128 to +127

**(2) Part d)** They are accessible everywhere

**(2) Part e)** In registers or on stack

**(15) Question 2.**

**Check: LDR R2,=X //pointer to X**

**MOVS R3,#0**

**LDRSB R1,[R2,R3] //value of X**

**CMP R1,#10 //**

**BLE zero //signed branch**

**MOVS R0,#1 //return 1**

**B out**

**zero: MOVS R0,#0 //return 0**

**out: BX LR**

**(30) Question 3.** You are passed two arrays

**int32\_t Find(int32\_t Buf1[],uint32\_t n1,**

**int32\_t Buf2[],uint32\_t n2){**

**// double nested loop, using indexing**

**for(uint32\_t i1=0; i1<n1; i1++){**

**for(uint32\_t i2=0; i2<n2; i2++){**

**if(Buf1[i1] == Buf2[i2]) return Buf1[i1];**

**}**

**}**

**return 0x80000000;**

**}**

**// double nested loop, using pointers**

**for(;n1 ; n1--){**

**uint32\_t n=n2;**

**int32\_t \*p=Buf2;**

**for(; n; n--){**

**if(\*Buf1 == \*p) return \*p;**

**p++;**

**}**

**Buf1++;**

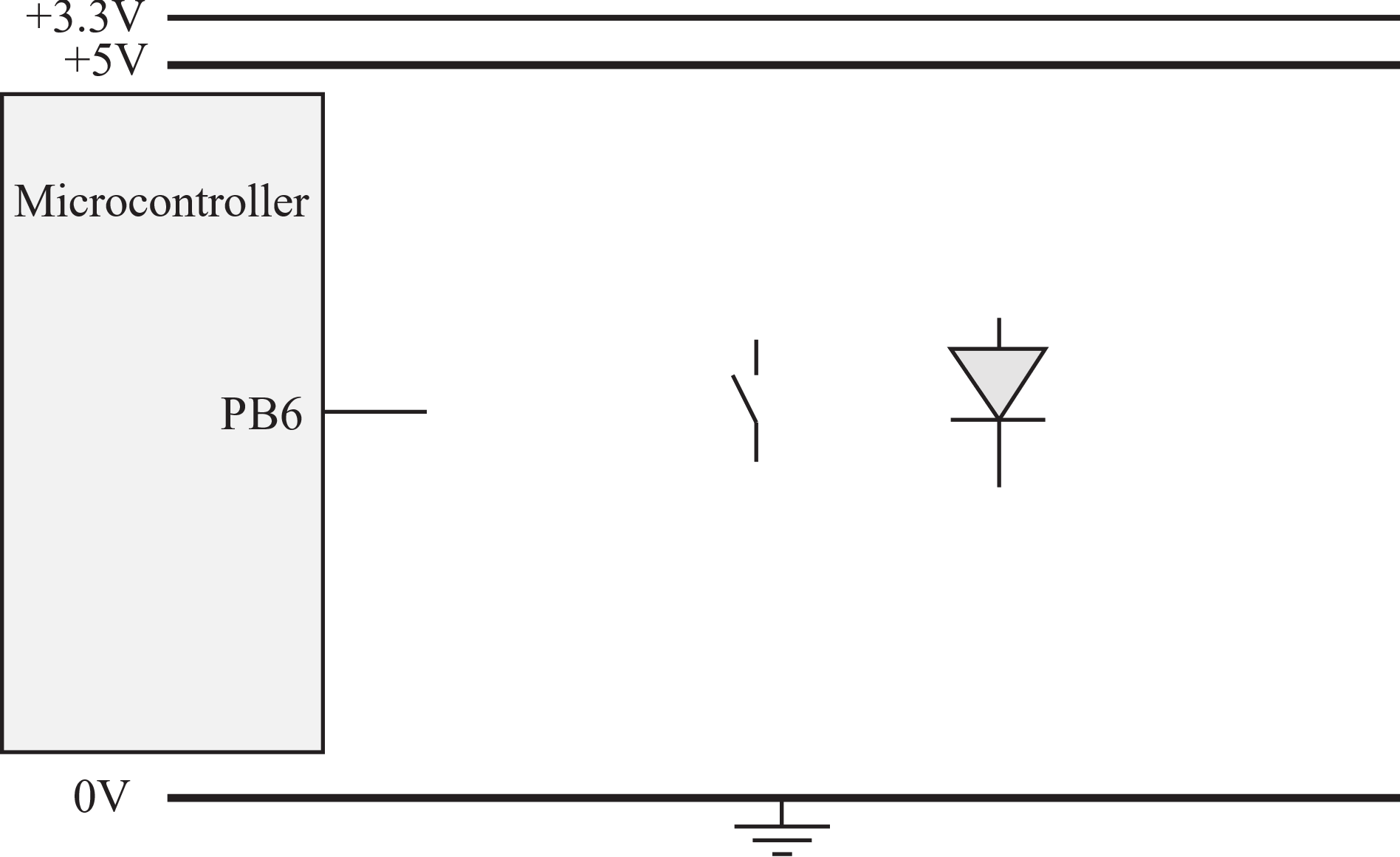
**}**

**return 0x80000000;**

**}**

**(13) Question 4.** Interface the switch to Port B using negative logic.

**(5) Part a)** R = 10k.



**(8) Part 6)**

**Switch\_Input:**

**LDR R1,=GPIOB\_DIN31\_0 // input register**

**LDR R0,[R1] // all of input**

**MOVS R2,#0x40 // bit 6 mask**

**ANDS R0,R0,R2 // just PB6**

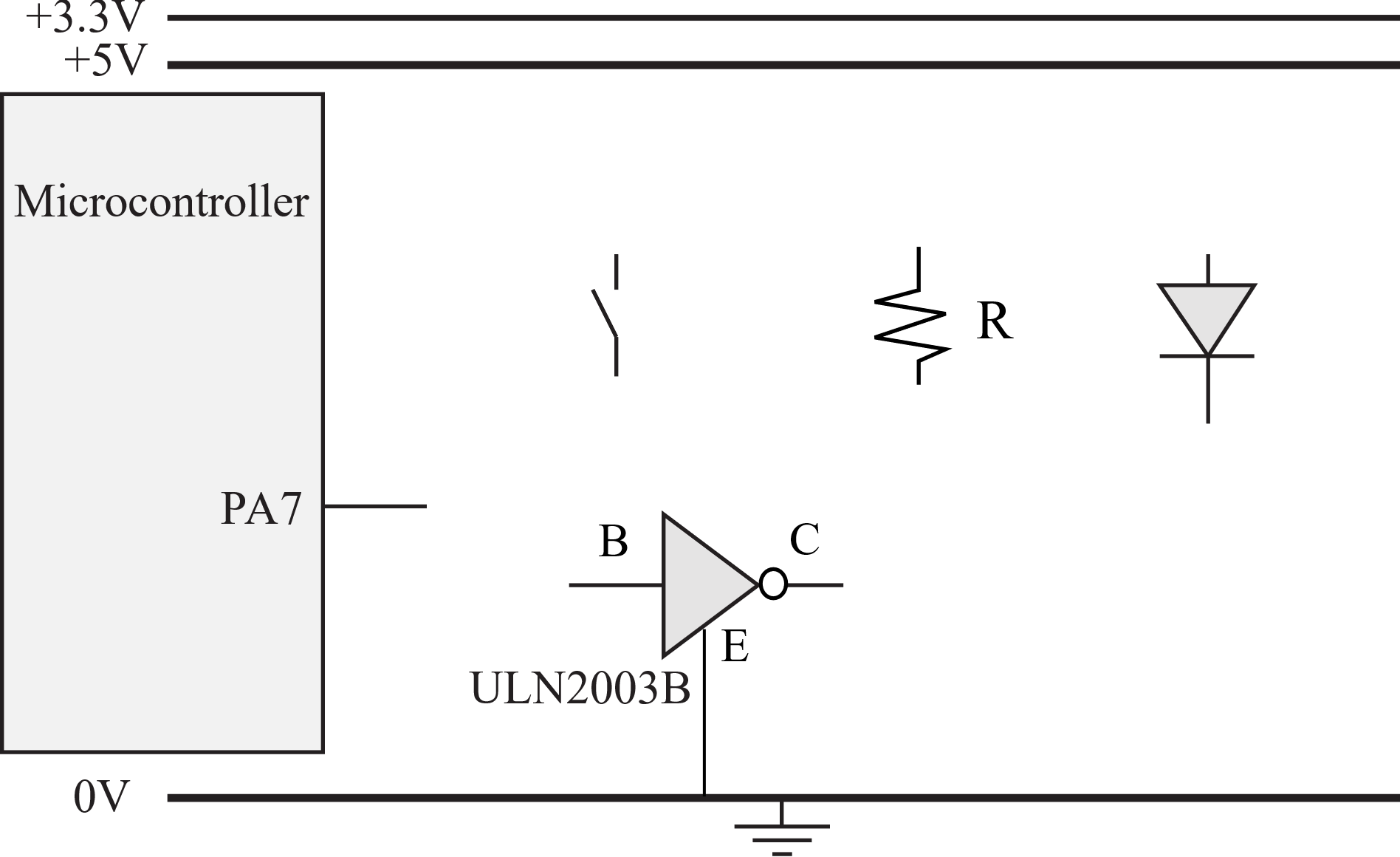
**EORS R0,R0,R2 // now positive logic**

**LSRS R0,R0,#6 // R0 is 0 or 1**

**BX LR**

**(12) Question 5.** Interface an LED to Port A using positive logic.

**(5) Part a)**



R =(3.0-1.6V)/2mA

= 1.4V/2mA

=1400mV/2mA

=700 ohms

**(7) Part b)** The software must be friendly.

**void LED\_Out(uint32\_t n){**

**if(n){**

**GPIOB->DOUTSET31\_0 = 1<<7;**

**else{**

**GPIOB->DOUTCLR31\_0 = 1<<7;**

**}**

**}**

**(5) Question 6.**

R4 = 5

**PUSH {R4,R6}**

**SP-> 4**

**6**

R5 = 30

**PUSH {R7,R5}//order doesn’t matter**

**SP-> 5**

R6 = 4

**7**

**4**

**6**

R7 = 6

**MULS R5,R5,R6 //R5=5\*6=30**

**POP {R6,R4} //R4=5, R6=7**

**SP-> 4**

**6**

**POP {R7,R6} //R6=4, R7=6**

**(5) Question 7.** 16-bit store will ignore the top 16 bits of R3, yielding just 0xC0D0. The 16-bit data 0xC0D0 is stored little endian.

|  |  |  |
| --- | --- | --- |
| Address | Contents |  |
| 0x20201000 |  | **<- R2** |
| 0x20201001 |  |  |
| 0x20201002 | **0xD0** | **<- R2+2** |
| 0x20201003 | **0xC0** |  |
| 0x20201004 |  |  |
| 0x20201005 |  |  |

**DCB 1,2,3 ; allocates three 8-bit byte(s)**

**DCW 1,2,3 ; allocates three 16-bit halfwords**

**DCD 1,2,3 ; allocates three 32-bit words**

**SPACE 4 ; reserves 4 bytes**