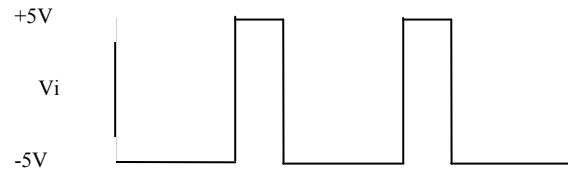
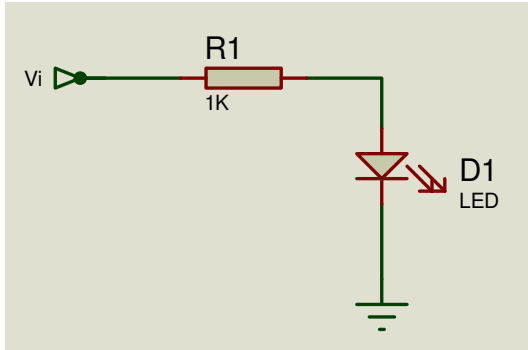


Nom i Cognoms: _____ **Una possible Solució** _____

1) (2p.) Apliquem el senyal Vi a l'entrada del circuit que es mostra a la figura. El senyal Vi té un duty cycle del 25%.



Considereu $V_\gamma = 2V$.

a) Quin és el corrent que passa pel led quan aquest està encès?

$$I_L = (V_i - V_\gamma) / R_1 = (5-2) / 1K$$

$$I_L = 3 \text{ mA}$$

b) Quin és el consum mig del led?

$$P_L = 0.25 (V_\gamma * I_L) = 0.25 (2V * 3mA) = 1.5mW$$

$$P_L = 1.5mW$$

c) Quin és el consum mig de la resistència?

$$P_R = 0.25 (V_R * I_L) = 0.25 (3V * 3mA) = 2.25mW$$

$$P_R = 2.25mW$$

d) Quina és la potència produïda pel generador de funcions?

$$P_{GF} = 0.25 (V_i * I_L) = 0.25 (5V * 3mA) = 3.75mW$$

$$P_{GF} = 3.75mW$$

2) (1p.) Quina és l'amplada en bits de la pila del 18F4550 ? Per quin motiu ?

20 bits. Ha de guardar les adreces de retorn, i l'espai d'adreçament és de 2^{20} instruccions.

3) (1p.) Volem calcular la XOR de les N primeres variables ubicades a memòria ($1 < N < 50$) i deixar el resultat a la posició 0x40.

Completeu l'esquelet del programa següent en llenguatge ensamblador del PIC18F4550.

```
#include "p18F4550.inc"
```

```
Result EQU 0x40
```

```
N EQU 0x41
```

```
Org 0
```

```
...
```

```
... ; inicialització de N i codi correcte però irrellevant
```

```
...
```

```
; escriviu el vostre codi a partir d'aquí
```

```
MOVLW 0 ; esborrem W
```

```
LFSR 0,0 ; inicialitzem FSR0 amb 0. Apuntador a la 1ª variable
```

```
Loop
```

```
XORWF POSTINC0,0,0 ; XOR de W amb la variable apuntada. Incrementem l'apuntador
```

```
DECFSZ N,1,0 ; decrementem N
```

```
Bra Loop
```

```
MOVWF Result,0 ; guardem el resultat
```

```
; a partir d'aquest punt només hi ha codi correcte però irrellevant
```

```
...
```

```
...
```

```
...
```

```
END
```

4) (1p.) Cita dos avantatges i dos inconvenients d'un sistema d'E/S mapejat a memòria respecte d'un que no ho estigui.

Avantatges: No calen senyals específics en el bus de control per a diferenciar accessos a memòria o E/S. Joc d'instruccions més reduït, i per tant, arquitectura més senzilla.

Inconvenients: Les instruccions no diferencien entre accés a memòria o E/S, això fa la programació més confusa i el control de memòria més ferragós. Els ports d'E/S ocupen part de la memòria disponible per a dades.

Nom i Cognoms: _____ **Una possible Solució** _____

5) El següent codi fa la divisió (entera) d'un nombre positiu representat amb 8 bits guardat a la posició 000h entre un nombre positiu representat amb 8 bits guardat a la posició 001h deixant el quocient a la posició 002h.

	TSTFSZ	01,0	
	BRA	cont	
	GOTO	error	;Si el divisor és 0, error
cont:	CLRF	02,0	
	MOVF	01,0,0	
bucle:	SUBWF	00,1,0	
	BN	final	;Si la resta no dóna negatiu seguirem
	INCF	02,1,0	;Incrementem el resultat
	BRA	bucle	
final:	...		;En aquest punt tenim el resultat

a) Quants cicles triga a fer la divisió si @000h=100d i @001h=11d ? (1,5 p.)

Hi haurà una part inicial amb TSTFSZ (1 cicle), BRA (2 cicles), CLRF (1 cicle) i MOVF (1 cicle) = 5 cicles. Després s'executa el bucle.

El bucle consta, quan no salta a final (no negatiu), de les instruccions:

SUBWF (1 cicle), BN (1 cicle), INCF (1 cicle), BRA (2 cicles) = 5 cicles.

En el darrer cas, el que la resta dóna negatiu i per tant salta a final, executa:

SUBWF (1 cicle), BN (2 cicles) = 3 cicles.

Quan dividim 100 entre 11, tenim 9 execucions de bucle amb (000) inicial a = 100,89,78,67,56,45,34,23,12 que faran 5 cicles i 1 execució amb (000) inicial a 1 que executarà 3 cicles.

Per tant el codi trigarà:

5 cicles (inicial) + 9 x 5 cicles (bucle que no surt a final) + 3 cicles (bucle que surt a final) = 53 cicles d'instrucció o 212 cicles de rellotge. Si tenim en compte el fetch de la primera instrucció 54 i 216.

b) Quants cicles trigarà en el cas pitjor (1 p.)?

El cas pitjor serà quan dividim 127 entre 1. En aquest cas trigarà:

5 cicles (inicial) + 127 cicles sense saltar x 5 + 3 cicles del darrer cas = 643 cicles d'instrucció (644 amb el primer fetch).

El número 255 (FF h) es codificaria com a negatiu i l'execució no seria l'esperada.

Nota. Es considerarà el cas 255 de cara a la correcció.

c) Quants Bytes ocupa a memòria el codi anterior (1 p.)?

Cada instrucció single-word ocupa 2 Bytes. Les instruccions double-word (en aquest cas hi ha el GOTO) ocupen 4 Bytes. Per tant en aquest cas tenim 2 + 2 + 4 + 2 + 2 + 2 + 2 + 2 + 2 = 20 Bytes.

6) En les instruccions que utilitzen un literal, en quina memòria (codi / dades) està gravat el literal? Posa un exemple de com es codificaria una instrucció que utilitzi un literal (0,5 p.)

Els literals es codifiquen a memòria de codi, juntament amb el codi d'operació. Per exemple, la instrucció ANDLW que fa la *and* bit a bit d'un literal i el Working Register es codificaria:

ANDLW 35h -> 0000101100110101

7) Explica raonadament com queden les posicions 00 del bank0 i del bank1 després d'executar aquest codi (1 p.)

MOVLB	1	;Posem el BSR a 1	
MOVLW	5	;Posem el WREG a 5	> WREG = 0
CLRF	00,1	;Posem l'adreça 100 a 0 (banked)	> (100) = 0
CLRF	00,0	;Posem l'adreça 000 a 0 (access)	> (000) = 0
ADDWF	00,1,1	;Sumem 5 a l'adreça 100 (banked i destí és file)	> (100) = 5
SUBWF	00,0,0	;Restem 5 del contingut de l'adreça 000 i guarda al W	> WREG = FBh
ADDWF	00,1,1	;Sumem el WREG amb el que hi ha a 100 i guardem a 100	> (100) = 0

Per tant: (000) = 0, (100) = 0

Mnemonic, Operands	Description	Cycles	16-Bit Instruction Word				Status Affected	Notes	
			MSb		LSb				
BYTE-ORIENTED OPERATIONS									
ADDWF	f, d, a	Add WREG and f	1	0010	01da	ffff	ffff	C, DC, Z, OV, N	1, 2
ADDWFC	f, d, a	Add WREG and Carry bit to f	1	0010	00da	ffff	ffff	C, DC, Z, OV, N	1, 2
ANDWF	f, d, a	AND WREG with f	1	0001	01da	ffff	ffff	Z, N	1, 2
CLRF	f, a	Clear f	1	0110	101a	ffff	ffff	Z	2
COMF	f, d, a	Complement f	1	0001	11da	ffff	ffff	Z, N	1, 2
CPFSEQ	f, a	Compare f with WREG, Skip =	1 (2 or 3)	0110	001a	ffff	ffff	None	4
CPFSGT	f, a	Compare f with WREG, Skip >	1 (2 or 3)	0110	010a	ffff	ffff	None	4
CPFSLT	f, a	Compare f with WREG, Skip <	1 (2 or 3)	0110	000a	ffff	ffff	None	1, 2
DECF	f, d, a	Decrement f	1	0000	01da	ffff	ffff	C, DC, Z, OV, N	1, 2, 3, 4
DECFSZ	f, d, a	Decrement f, Skip if 0	1 (2 or 3)	0010	11da	ffff	ffff	None	1, 2, 3, 4
DCFSNZ	f, d, a	Decrement f, Skip if Not 0	1 (2 or 3)	0100	11da	ffff	ffff	None	1, 2
INCF	f, d, a	Increment f	1	0010	10da	ffff	ffff	C, DC, Z, OV, N	1, 2, 3, 4
INCFSZ	f, d, a	Increment f, Skip if 0	1 (2 or 3)	0011	11da	ffff	ffff	None	4
INFSNZ	f, d, a	Increment f, Skip if Not 0	1 (2 or 3)	0100	10da	ffff	ffff	None	1, 2
IORWF	f, d, a	Inclusive OR WREG with f	1	0001	00da	ffff	ffff	Z, N	1, 2
MOVF	f, d, a	Move f	1	0101	00da	ffff	ffff	Z, N	1
MOVFF	f _s , f _d	Move f _s (source) to f _d (destination) 1st word	2	1100	ffff	ffff	ffff	None	
		f _d (destination) 2nd word		1111	ffff	ffff	ffff		
MOVWF	f, a	Move WREG to f	1	0110	111a	ffff	ffff	None	
MULWF	f, a	Multiply WREG with f	1	0000	001a	ffff	ffff	None	1, 2
NEGF	f, a	Negate f	1	0110	110a	ffff	ffff	C, DC, Z, OV, N	
RLCF	f, d, a	Rotate Left f through Carry	1	0011	01da	ffff	ffff	C, Z, N	1, 2
RLNCF	f, d, a	Rotate Left f (No Carry)	1	0100	01da	ffff	ffff	Z, N	
RRCF	f, d, a	Rotate Right f through Carry	1	0011	00da	ffff	ffff	C, Z, N	
RRNCF	f, d, a	Rotate Right f (No Carry)	1	0100	00da	ffff	ffff	Z, N	
SETF	f, a	Set f	1	0110	100a	ffff	ffff	None	1, 2
SUBFWB	f, d, a	Subtract f from WREG with Borrow	1	0101	01da	ffff	ffff	C, DC, Z, OV, N	
SUBWF	f, d, a	Subtract WREG from f	1	0101	11da	ffff	ffff	C, DC, Z, OV, N	1, 2
SUBWFB	f, d, a	Subtract WREG from f with Borrow	1	0101	10da	ffff	ffff	C, DC, Z, OV, N	
SWAPF	f, d, a	Swap Nibbles in f	1	0011	10da	ffff	ffff	None	4
TSTFSZ	f, a	Test f, Skip if 0	1 (2 or 3)	0110	011a	ffff	ffff	None	1, 2
XORWF	f, d, a	Exclusive OR WREG with f	1	0001	10da	ffff	ffff	Z, N	

Nom i Cognoms: Una possible Solució

Mnemonic, Operands	Description	Cycles	16-Bit Instruction Word				Status Affected	Notes
			MSb		LSb			
LITERAL OPERATIONS								
ADDLW	k	Add Literal and WREG	1	0000	1111	kkkk	kkkk	C, DC, Z, OV, N
ANDLW	k	AND Literal with WREG	1	0000	1011	kkkk	kkkk	Z, N
IORLW	k	Inclusive OR Literal with WREG	1	0000	1001	kkkk	kkkk	Z, N
LFSR	f, k	Move Literal (12-bit) 2nd word to FSR(f) 1st word	2	1110	1110	00ff	kkkk	None
MOVLB	k	Move Literal to BSR<3:0>	1	0000	0001	0000	kkkk	None
MOVLW	k	Move Literal to WREG	1	0000	1110	kkkk	kkkk	None
MULLW	k	Multiply Literal with WREG	1	0000	1101	kkkk	kkkk	None
RETLW	k	Return with Literal in WREG	2	0000	1100	kkkk	kkkk	None
SUBLW	k	Subtract WREG from Literal	1	0000	1000	kkkk	kkkk	C, DC, Z, OV, N
XORLW	k	Exclusive OR Literal with WREG	1	0000	1010	kkkk	kkkk	Z, N
DATA MEMORY ↔ PROGRAM MEMORY OPERATIONS								
TBLRD*	Table Read	2	0000	0000	0000	1000	None	
TBLRD*+	Table Read with Post-Increment		0000	0000	0000	1001	None	
TBLRD*-	Table Read with Post-Decrement		0000	0000	0000	1010	None	
TBLRD*+	Table Read with Pre-Increment		0000	0000	0000	1011	None	
TBLWT*	Table Write	2	0000	0000	0000	1100	None	
TBLWT*+	Table Write with Post-Increment		0000	0000	0000	1101	None	
TBLWT*-	Table Write with Post-Decrement		0000	0000	0000	1110	None	
TBLWT*+	Table Write with Pre-Increment		0000	0000	0000	1111	None	

Mnemonic, Operands		Description	Cycles	16-Bit Instruction Word				Status Affected	Notes
				MSb		LSb			
BIT-ORIENTED OPERATIONS									
BCF	f, b, a	Bit Clear f	1	1001	bbba	ffff	ffff	None	1, 2
BSF	f, b, a	Bit Set f	1	1000	bbba	ffff	ffff	None	1, 2
BTFSC	f, b, a	Bit Test f, Skip if Clear	1 (2 or 3)	1011	bbba	ffff	ffff	None	3, 4
BTFSS	f, b, a	Bit Test f, Skip if Set	1 (2 or 3)	1010	bbba	ffff	ffff	None	3, 4
BTG	f, d, a	Bit Toggle f	1	0111	bbba	ffff	ffff	None	1, 2
CONTROL OPERATIONS									
BC	n	Branch if Carry	1 (2)	1110	0010	nnnn	nnnn	None	4
BN	n	Branch if Negative	1 (2)	1110	0110	nnnn	nnnn	None	
BNC	n	Branch if Not Carry	1 (2)	1110	0011	nnnn	nnnn	None	
BNN	n	Branch if Not Negative	1 (2)	1110	0111	nnnn	nnnn	None	
BNOV	n	Branch if Not Overflow	1 (2)	1110	0101	nnnn	nnnn	None	
BNZ	n	Branch if Not Zero	1 (2)	1110	0001	nnnn	nnnn	None	
BOV	n	Branch if Overflow	1 (2)	1110	0100	nnnn	nnnn	None	
BRA	n	Branch Unconditionally	2	1101	0nnn	nnnn	nnnn	None	
BZ	n	Branch if Zero	1 (2)	1110	0000	nnnn	nnnn	None	
CALL	n, s	Call Subroutine 1st word	2	1110	110s	kkkk	kkkk	None	
		2nd word		1111	kkkk	kkkk	kkkk		
CLRWDI	—	Clear Watchdog Timer	1	0000	0000	0000	0100	TO, PD	
DAW	—	Decimal Adjust WREG	1	0000	0000	0000	0111	C	
GOTO	n	Go to Address 1st word	2	1110	1111	kkkk	kkkk	None	
		2nd word		1111	kkkk	kkkk	kkkk		
NOP	—	No Operation	1	0000	0000	0000	0000	None	
NOP	—	No Operation	1	1111	xxxx	xxxx	xxxx	None	
POP	—	Pop Top of Return Stack (TOS)	1	0000	0000	0000	0110	None	
PUSH	—	Push Top of Return Stack (TOS)	1	0000	0000	0000	0101	None	
RCALL	n	Relative Call	2	1101	1nnn	nnnn	nnnn	None	
RESET		Software Device Reset	1	0000	0000	1111	1111	All	
RETFIE	s	Return from Interrupt Enable	2	0000	0000	0001	000s	GIE/GIEH, PEIE/GIEL	
RETLW	k	Return with Literal in WREG	2	0000	1100	kkkk	kkkk	None	
RETURN	s	Return from Subroutine	2	0000	0000	0001	001s	None	
SLEEP	—	Go into Standby mode	1	0000	0000	0000	0011	TO, PD	

TABLE 5-2: REGISTER FILE SUMMARY

File Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on POR, BOR	Details on page
TOSU	—	—	—	Top-of-Stack Upper Byte (TOS<20:16>)					---0 0000	53, 60
TOSH	Top-of-Stack High Byte (TOS<15:8>)								0000 0000	53, 60
TOSL	Top-of-Stack Low Byte (TOS<7:0>)								0000 0000	53, 60
STKPTR	STKFUL	STKUNF	—	SP4	SP3	SP2	SP1	SP0	00-0 0000	53, 61
PCLATU	—	—	—	Holding Register for PC<20:16>					---0 0000	53, 60
PCLATH	Holding Register for PC<15:8>								0000 0000	53, 60
PCL	PC Low Byte (PC<7:0>)								0000 0000	53, 60
TBLPTRU	—	—	bit 21 ⁽¹⁾	Program Memory Table Pointer Upper Byte (TBLPTR<20:16>)					--00 0000	53, 84
TBLPTRH	Program Memory Table Pointer High Byte (TBLPTR<15:8>)								0000 0000	53, 84
TBLPTRL	Program Memory Table Pointer Low Byte (TBLPTR<7:0>)								0000 0000	53, 84
TABLAT	Program Memory Table Latch								0000 0000	53, 84
PRODH	Product Register High Byte								XXXXXXXX XXXXXX	53, 97
PRODL	Product Register Low Byte								XXXXXXXX XXXXXX	53, 97
INTCON	GIE/GIEH	PEIE/GIEL	TMR0IE	INT0IE	RBIE	TMR0IF	INT0IF	RBIF	0000 000x	53, 101
INTCON2	RBPV	INTEDG0	INTEDG1	INTEDG2	—	TMR0IP	—	RBIP	1111 -1-1	53, 102
INTCON3	INT2IP	INT1IP	—	INT2IE	INT1IE	—	INT2IF	INT1IF	11-0 0-00	53, 103
INDF0	Uses contents of FSR0 to address data memory – value of FSR0 not changed (not a physical register)								N/A	53, 75
POSTINC0	Uses contents of FSR0 to address data memory – value of FSR0 post-incremented (not a physical register)								N/A	53, 76
POSTDEC0	Uses contents of FSR0 to address data memory – value of FSR0 post-decremented (not a physical register)								N/A	53, 76
PREINC0	Uses contents of FSR0 to address data memory – value of FSR0 pre-incremented (not a physical register)								N/A	53, 76
PLUSW0	Uses contents of FSR0 to address data memory – value of FSR0 pre-incremented (not a physical register) – value of FSR0 offset by W								N/A	53, 76
FSR0H	—	—	—	—	Indirect Data Memory Address Pointer 0 High Byte				---- 0000	53, 75
FSR0L	Indirect Data Memory Address Pointer 0 Low Byte								XXXXXXXX XXXXXX	53, 75
WREG	Working Register								XXXXXXXX XXXXXX	53
INDF1	Uses contents of FSR1 to address data memory – value of FSR1 not changed (not a physical register)								N/A	53, 75
POSTINC1	Uses contents of FSR1 to address data memory – value of FSR1 post-incremented (not a physical register)								N/A	53, 76
POSTDEC1	Uses contents of FSR1 to address data memory – value of FSR1 post-decremented (not a physical register)								N/A	53, 76
PREINC1	Uses contents of FSR1 to address data memory – value of FSR1 pre-incremented (not a physical register)								N/A	53, 76
PLUSW1	Uses contents of FSR1 to address data memory – value of FSR1 pre-incremented (not a physical register) – value of FSR1 offset by W								N/A	53, 76
FSR1H	—	—	—	—	Indirect Data Memory Address Pointer 1 High Byte				---- 0000	53, 75
FSR1L	Indirect Data Memory Address Pointer 1 Low Byte								XXXXXXXX XXXXXX	53, 75
BSR	—	—	—	—	Bank Select Register				---- 0000	54, 65

