

Nom i Cognoms: \_\_\_\_\_ POSSIBLE SOLUCIÓ \_\_\_\_\_

## Computer Interfacing. Primer parcial, 9/10/2017

Responen les preguntes a l'espai reservat. Justifiqueu les respostes.

Poseu el nom a tots els fulls. Totes les preguntes valen 1 punt.

(Recordeu, Destí: W=WORKING, F= FILE, Bank: A=ACCESS, B=BSR)

1. Volem ampliar la memòria de programa del PIC184550 de 32K a 128K. Quines modificacions haurem de fer a l'arquitectura del PIC per tal de que tot funcioni correctament?

No haurem de fer cap modificació. L'espai d'adreces de memòria de programa es de 2MB i admet memòries de programa de fins aquest tamany sense haver de fer cap modificació a l'arquitectura del PIC.

2. El següent programa en *assembler* del PIC18F4550 implementa una suma de dos enters de 24 bits emmagatzemats en les posicions de memòria 0x10-0x12 i 0x13-0x15 respectivament i guarda el resultat en la posicions de memòria 0x20-0x22 . Emplena els forats que falten.

```
#include <p18F8720.inc>
ORG 0x00
GOTO START
ORG 0x08
RETFIE
ORG 0x18
RETFIE
start: MOVF 0x10 , W, A
ADDWF 0x13, W, A
MOVWF 0x20 , A
MOVF 0x11 , W, A
ADDWFC 0x14 , W, A
MOVWF 0x21 , A
MOVF 0x12 , W, A
ADDWFC 0x15 , W, A
MOVWF 0x22, A
End
```

3. Calcula de temps que triga en executar-se la següent subrutina en un PIC18F4550 funcionant amb un oscil·lador a 4MHz.

```
Delay:
MOVLW d'250'
Loop:
ADDLW 0xFF
BTFFSS STATUS, Z
BRA Loop
RETURN
```

Tenim que el període del CLK és de  $1/(4 \cdot 10^6)$  per tant cada cicle d'instrucció serà de 1us (1 microsegon)

El primer MOVLW triga un cicle i s'executa un cop

ADDLW triga un cicle i s'executa 250 vegades.

BTFFSS triga un cicle quan no es compleix condició 249 vegades

Triga dos cicles quan es compleix condició

BRA dos cicles 249 vegades

RETURN 2 cicles una vegada.

Total =  $1 + 249 \cdot (1 + 1 + 2) + (1 + 2) + 2 = 1002$  cicles = 1002 us

Nom i Cognoms: \_\_\_\_\_

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 <sub>s</sub> , f <sub>d</sub>	Move f <sub>s</sub> (source) to f <sub>d</sub> (destination) 1st word	2	1100	ffff	ffff	ffff	None	
				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	

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	
				1111	0000	kkkk	kkkk		
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		

Nom i Cognoms: \_\_\_\_\_

4. Quines dues instruccions del PIC18F utilitzaríeu per passar dades de la ROM a la RAM?

Qualsevol instrucció que faci ús dels literals (els literals estan emmagatzemats en el codi de la ROM). Per exemple `MOVLW VALUE` i posteriorment `MOVF DESTINATION`.

O bé utilitzar la instrucció específica per llegir de la ROM: `TBLRD`

5. Convertiu a C el següent programa fet en *assembler*:

```
          CLR_F _x, A
          CLR_F _i, A

Loop:     INC_F  _i, F, A
          MOV_F  _i, W, A
          ADDWF  _x, F, A
          MOVLW  9
          CPFSGT _i, A
          GOTO   Loop
```

El programa calcula el valor de la suma dels deu primers enters no nuls, obtenint el resultat en la variable `_x` = 1+2+3+4+5+6+7+8+9+10 = 55. Colateralment la variable `_i` s'incrementa fins a 10.

El codi que més s'aproxima a l'indicat és:

```
_x = 0;
_i = 0;
do {
    _i = _i+1;
    _x = _x + i;
} while( _i <= 9)
```

6. Quina funció implementa el següent codi?

```
          MOVLW      0
          MOVFF  _var1, _var2
          ADDWF   _var1, W, A
          BNN     final
          NEGF     _var2, A

final:     ...
```

El que fa és: `_var2 = abs(_var1)`

Nom i Cognoms: \_\_\_\_\_

**TABLE 26-2: PIC18FXXXX INSTRUCTION SET (CONTINUED)**

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		
CLRWDT	—	Clear Watchdog Timer	1	0000	0000	0000	0100	$\overline{TO}$ , $\overline{PD}$	
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	$\overline{TO}$ , $\overline{PD}$	

Nom i Cognoms: \_\_\_\_\_

7. El mode Banked i l'Access Bank són necessaris perquè a la majoria d'instruccions tenim 8 bits per codificar el paràmetre (file) i la memòria de dades s'ha d'adreçar amb 12 bits.

Què caldria canviar a l'arquitectura del PIC per fer que totes les instruccions existents puguin tenir un paràmetre de 12 bits (no volem modificar el joc d'instruccions).

Marca amb un cercle **totes** les opcions certes (cada encert suma 0,5, cada error resta 0,33):

☒ (a) l'amplada de l'Instruction Register (IR) (ara només té 16 bits).

☐ (b) l'amplada de la pila del sistema (ara només té 20 bits).

☐ (c) el nombre de nivells de la pila del sistema (ara només té 31 nivells).

☒ (d) la mida de paraula de la memòria de programa (ara només té 16 bits).

☐ (e) la freqüència del processador (al tenir més bits d'adreçament pot anar més ràpid).

Si ampliem el paràmetre de la instrucció en 4 bits, la memòria de programa haurà de ser més ampla (20 bits) i l'IR també (conté el que ve de memòria de programa, també). La pila no es veurà afectada (guarda adreces no instruccions) i el clock de la CPU no està relacionat.

8. Perquè és aconsellable fer servir la instrucció BRA en comptes del GOTO pels salts incondicionals on l'adreça de destí sigui propera?

Degut a que ocupa només una paraula a memòria (2B), el GOTO ocupa dues paraules (4B). El temps d'execució és de dos cicles per cada una. En el cas de que tinguem un skip davant i s'activi, el GOTO provocarà que l'skip trigui 3 cicles i en canvi el BRA 2.

9. En el mode indirecte ens cal un registre especial per fer d'apuntador a la memòria de dades. Com podem posar un valor a aquest registre per inicialitzar aquest punter?

Aquest registre és el File Select Register i en tenim 3: FSR0, FSR1 i FSR2. Per donar-li un valor, el processador disposa de la instrucció LFSR (load FILE SELECT REGISTER) que té dos paràmetres; l'índex del FSR (0-2) i el valor de 12 bits del punter a memòria de dades.

10. Perquè en el joc d'instruccions del PIC hi ha dues instruccions NOP; una codificada amb tot 0 i l'altra codificada amb quatre 1 a l'inici?

El NOP 0000 0000 0000 0000 és la codificació de la instrucció NOP que s'utilitza durant la programació normal del micro i que permet que si el Program Counter surt de la memòria de programa implementada (un adreçament fora del rang torna sempre 0s) no s'executi res.

El NOP 1111 .... .... es fa servir a la codificació de la segona paraula de les instruccions Double Word (GOTO, CALL, MOVFF, LFSR). Si degut a un skip el Program Counter arriba a la seva adreça no s'executarà res.