II salti

CONTROL OPERATIONS

BC	n	Branch if Carry	1 (2)	1110	0010	nnnn	nnnn	None
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
GOTO	k	Go to address 1st word	2	1110	1111	kkkk	kkkk	None
		2nd word		1111	kkkk	kkkk	kkkk	

I programmi difficilmente sono costituiti da una semplice sequenza di istruzioni da eseguire una dopo l'altra: in genere sono presenti cicli o diramazioni, cioè esecuzione non sequenziale di istruzioni.

L'indirizzo dell'istruzione che verrà eseguite è contenuta ne Program Counter PC. Esso potrà:

- Cambiare "automaticamente", incrementandosi di uno al termine di ciascuna istruzione. In questo modo le istruzioni vengono eseguite sequenzialmente
- Aumentare di due, in pratica saltando l'istruzione immediatamente successiva
- Assumere un nuovo valore, indipendente dal precedente: *salto assoluto*
- Assumere un nuovo valore, sommando o sottraendo un numero al valore attuale del PC: *salto relativo*
- Non cambiare, per esempio dopo aver eseguito l'istruzione `sleep` (e quindi fermare l'esecuzione di qualunque programma)

Inoltre il salto può essere fatto sempre (*salto incondizionato*) oppure solo al manifestarsi di determinate condizioni, tipicamente il valore di un flag.

Salto incondizionato

Le istruzioni di salto incondizionato permette di eseguire una istruzione posta in una qualunque cella della memoria programma, *saltando* in avanti o indietro all'interno del programma. Il nuovo indirizzo da porre nel PC può essere scritto direttamente nel codice come numero oppure, cosa assolutamente preferibile, fatta calcolare automaticamente all'assemblatore attraverso l'uso di una *label*.

Esempio - Il codice seguente esegue infinite volte l'incremento unitario del registro `WREG` che passa dal valore iniziale 0 a 255 per poi tornare a 0 per ricominciare nuovamente:

```
        clrf WREG
ripeti  incf WREG
        goto ripeti
```

Il programma, una volta caricato nella memoria flash, appare nel seguente modo:

Line	Address	Opcode	Label	DisAssy
1	0000	6AE8		CLRF WREG, ACCESS
2	0002	2AE8		INCF WREG, F, ACCESS
3	0004	EF01		GOTO 0x2
4	0006	F000		NOP

Si noti in particolare come l'istruzione `goto ripeti` sia stata assemblata come `goto 0x2`. Questo numero è costituito da 21 bit:

- `0000 0000 0000 0000 0001` (`0x00001`): le cinque cifre esadecimali evidenziate in rosso nell'immagine (20 bit)
- un ventunesimo bit, fisso a zero, da aggiungere a destra (LSB). Quindi l'indirizzo a cui viene impostato il PC è, in binario: `0 0000 0000 0000 0010` (cioè `0x000002` oppure `0x2`)

`0x2` è l'indirizzo a cui si trova l'istruzione `incf WREG` e la label `ripeti`.

Branch (diramazione) incondizionato

Questa istruzione, a partire dal nome, è molto simile alla precedente. Essa somma (o sottrae) al valore del Program Counter un numero compreso tra -1024 e 1023, permettendo al programma di andare avanti (o indietro) saltando fino ad un massimo di circa 1000 istruzioni rispetto all'istruzione stessa di `bra`.

A volte questo tipo di salto è indicato come *salto relativo*, mentre il `goto` è indicato come *salto assoluto*.

Esempio - Il codice seguente fa, apparentemente, l'identica cosa dell'istruzione *goto*:

```
        clrf WREG
ripeti  incf WREG
        bra ripeti
```

Il codice salvato in memoria è invece diverso (e più compatto):

Line	Address	Opcode	Label	DisAssy
1	0000	6AE8		CLRF WREG, ACCESS
2	0002	2AE8		INCF WREG, F, ACCESS
3	0004	D7FE		BRA 0x2

Facciamo riferimento al foglio tecnico e vediamo come è stato costruto:

BRA **Unconditional Branch**

Syntax:	BRA n
Operands:	$-1024 \leq n \leq 1023$
Operation:	$(PC) + 2 + 2n \rightarrow PC$
Status Affected:	None
Encoding:	1101 | 0nnn | nnnn | nnnn
Description:	Add the 2's complement number '2n' to the PC. Since the PC will have incremented to fetch the next instruction, the new address will be PC + 2 + 2n. This instruction is a two-cycle instruction.

Costruiamo a mano il codice operativo `0xD7FE (1101 0111 1111 1110)` dell'istruzione `bra ripeti`, mostrato nell'immagine:

- I primi 5 bit sono fissi: `1101 0`
- `0x7FE` (in binario: `111 1111 1110`) corrisponde al numero decimale negativo - 2 ed è indicato come `n` nei fogli tecnici; è espresso in complemento a due
- Il Program Counter a cui si trova l'istruzione `bra ripeti` è pari a `0x04`.
- In base alla formula riportata nel foglio tecnico, il nuovo `PC` è quindi pari a `PC + 2 + 2·n = 4 + 2 − 2·2 = 2`, indirizzo a cui è effettivamente posta la label `ripeti`.

Per fortuna è stato calcolato automaticamente dall'assemblatore...

Esercizio 2 - Eseguire il codice dell'esempio 2 passo-passo, osservando il valore del registro `WREG`

Branch condizionati

I salti condizionati permettono di eseguire un'istruzione piuttosto che un'altra a seconda del valore di uno dei *flags*. Sono alla base dei costrutti che ad alto livello vengono indicati come `if-then-else` oppure cicli `while-do` e `for`.

Per il test possono essere utilizzati i flag N, Z, C e OV, producendo 8 diverse istruzioni di salto (`bn, bnn, bz, bnz, bc, bnc bov e bnov`) a seconda se il salto debba avvenire quando flag vale zero oppure vale uno. Per la descrizione, peraltro implicita nel nome, si rimanda ai fogli tecnici e agli esempi di seguito riportati.

bnz

Questa istruzione esegue un salto in funzione del valore del flag Z.

Esempio 3 - Si consideri il seguente codice che fa riferimento al flag Z e all'istruzione `bnz` (**b**ranch if **n**ot **z**ero):

```
  movlw 0x10 ; Il numero 0x10 viene memorizzato in WREG
ripeti
  decf WREG  ; WREG viene decrementato di 1
  bnz ripeti ; Se il risultato non è zero, torna indietro
alla label ripeti
  sleep      ; Il programma termina
```

E' necessario prestare attenzione al fatto che il salto avviene o meno in corrispondenza del valore di un *flag* (nel caso specifico del flag ZERO) e non in funzione del contenuto di un registro.

BNZ	Branch if Not Zero
Syntax:	BNZ n
Operands:	$-128 \leq n \leq 127$
Operation:	if ZERO bit is '0' $(PC) + 2 + 2n \rightarrow PC$
Status Affected:	None
Description:	If the ZERO bit is '0', then the program will branch. The 2's complement number '2n' is added to the PC. Since the PC will have incremented to fetch the next instruction, the new address will be PC + 2 + 2n. This instruction is then a two-cycle instruction.

Example:

```
HERE    BNZ  Jump
```

Before Instruction
PC = address (HERE)
After Instruction
If ZERO = 0;
PC = address (Jump)
If ZERO = 1;
PC = address (HERE + 2)

Molto simile a `bnz`, l'istruzione `bz` (**B**ranch if **Z**ero), ovviamente dal funzionamento esattamente contrario.

Due osservazioni:

- La distanza tra l'istruzione `bnz` e la label deve essere minore di 128 istruzioni circa, in più o in meno
- Occorre prestare attenzione a quale è l'ultima istruzione che ha modificato il flag. Per esempio il seguente codice **non** funziona come ci di potrebbe aspettare:

```
ricomincia ; ATTENZIONE: questo codice non funziona come
sembra!
  movlw 0x10
  incf WREG
  movlw 0         ; Azzero WREG
  bnz ricomincia ; Se WREG non è zero, ricomincia
  sleep
```

bc e bnc

Esempio - Il codice seguente utilizza il flag C (carry) per eseguire un codice molto simile al precedente:

```
  movlw 0xF0
ancora
  incf WREG
  bnc ancora
finito
  sleep
```

Esercizio - Eseguire il codice passo-passo, osservando il valore del flag Z e del flag C

Confronti

Per eseguire il confronto tra una variabile ed un numero (o un'altra variabile) occorre eseguire due istruzioni:

- una sottrazione, ignorando il risultato. per esempio sublw (**SUB**tract **W** from **L**iteral)
- un salto, in relazione al valore del flag Z oppure C

Esempio 5: Il seguente codice confronta il contenuto del registro `W` con il numero `0x10` ed esegue alcune istruzioni se sono uguali oppure altre se sono diversi.

Con una sintassi *simil-C*: `if (WREG == 0x10) then ... else ...`

```
  movlw 0x30 ; Modificare il numero da mettere in WREG
  sublw 0x10 ; 0x10 - (WREG) -> WREG; WREG == 0x10 ?
  bz uguali
diversi ; Label non necessaria, solo per chiarezza
```

```
  nop ; Inserire qui le istruzioni da eseguire se WREG !=
0x10
  nop
  bra finito
uguali
  nop ; Inserire qui le istruzioni da eseguire se WREG ==
0x10
  nop
finito
  sleep
```

Esempio - Il seguente codice confronta il contenuto del registro `W` con il numero `0x10` ed esegue alcune istruzioni se `WREG` è maggiore di `0x10` oppure altre se minore o uguale.

Con una sintassi *simil-C*: `if (WREG > 0x10) then ... else ...`

```
  movlw 0x30 ; Modificare il numero da mettere in WREG
  sublw 0x10 ; 0x10 - (WREG) -> WREG; WREG == 0x10 ?
  bc minore
maggiore ; Label non necessaria, solo per chiarezza
  nop ; Inserire qui le istruzioni da eseguire se WREG >
0x10
  nop
  bra finito
minore
  nop ; Inserire qui le istruzioni da eseguire se WREG <=
0x10
  nop
finito
  sleep
```

Le istruzioni di skip

Un salto condizionato può essere anche ottenuto con diverse istruzioni di *skip*, traducibile come *salta l'istruzione successiva*. Spesso il codice prodotto è più intuitivo

Esempio - Decrementare WREG fino a 0, partendo da un valore qualunque (0x10 nel codice mostrato)

```
  movlw 0x10
ripeti
  decfsz WREG ; decrementa WREG e salta l'istruzione
seguente (bra sleep) se WREG == 0
  bra ripeti
  sleep
```