Answer Set 2

OUT: 13 OCTOBER, 1997

0.1 General remarks

We investigate which instructions in the PIC instruction are essential by finding code sequences which can replace unnecessary instructions. It is fairly easy to show replacements for almost all instructions if we don't care about what happens with the flags. However, the flags might be very important, so we should take care that the substitution affects the flags exactly as does the oiriginal instruction.

The situtation is even more difficult in the presence of interrupts. The code sequences often use a temporary variables. If an interrupt occurs in the middle of the code sequence, and the interrupt handler also uses the code sequence, the temporary variables will be overwritten. We could disable interrupts during the emulation, or we can have two versions of each replacement sequence, one for interrupt level and the other for non-interrupt level which use different register sets for temporary variables.

0.2 NOP

The NOP can be replaced with a GOTO the next instruction. Hence it is not necessary.

0.3 Barrel Shifting without SWAPF

Using right or left shifting, we can reduce the sequence to no more than 4 shifts. For instance, replace right shift 5 positions with left shift 3 positions. Here is left shift 3 on file register TARGET, with result in TARGET. The W register is preserved.

```
;
;   Barrel Shift Left 3 ( = Barrel Shift Right 5 )
;
    MOVWF    tempW    ; save W register
    RLF      TARGET,W ; set C flag correctly
    RLF      TARGET,F ; shift once
    RLF      TARGET,W ; set C flag correctly
    RLF      TARGET,F ; shift twice
    RLF      TARGET,W ; set C flag correctly
    RLF      TARGET,F ; shift three times
    MOVF     tempW,W  ; restore W register
;
```

0.4 SWAPF

SWAPF is a barrel shift left or right by 4. It is not necessary. The code is complicated by saving and restoring the W register and the Status.

```
;   SWAP     TARGET
;   no flags affected, W register preserved
```

```
MOVWF    tempW    ; save W register
BCF      tempStatus,C
BTFSC    STATUS,C
BSF      tempStatus,C  ; save C flag
BCF      tempStatus,Z
BTFSC    STATUS,Z
BSF      tempStatus,Z  ; save Z flag
RLF      TARGET,W ; set C flag correctly
RLF      TARGET,F ; shift once
RLF      TARGET,W ; set C flag correctly
RLF      TARGET,F ; shift twice
RLF      TARGET,W ; set C flag correctly
RLF      TARGET,F ; shift three times
RLF      TARGET,W ; set C flag correctly
RLF      TARGET,F ; shift three times
MOVF     tempW,W  ; restore W register
BCF      STATUS,C
BTFSC    tempStatus,C
BSF      STATUS,S ; restore C flag
BCF      STATUS,Z
BTFSC    tempStatus,Z
BSF      STATUS,Z ; restore Z flag
```

;

0.5 Barrel Shifting with SWAPF

SWAPF can be combined with RLF and RRF to give shorter barrel shifts. You can skip ahead 4 shifts with SWAPF, so that no more than two RLF or RRF stages are ever needed.

0.6 ANDWF

Using DeMorgan's Theorem, one of ANDWF or IORWF can be replaced with the other. Both are not needed.

```
;
;  Replacement for ANDWF TARGET,d, where d=0
    XORLW   OFF       ; complement W
    COMF    TARGET,F  ; complement the TARGET
    IORWF   TARGET,W  ; W has complement of W AND TARGET
    COMF    TARGET,F  ; restore TARGET
    XORLW   OFF       ; calculate W AND TARGET = NOT ( NOT W OR NOT TARGET )
                      ; and set Z flag correctly
;
```

```
 ;  Replacement where d=1
     XORLW   OFF       ; complement W
     COMF    TARGET,F  ; complement TARGET
     IORWF   TARGET,F  ; TARGET has complement of W AND TARGET
     XORLW   OFF       ; restore W
     COMF    TARGET    ; calculate W AND TARGET = NOT ( NOT W OR NOT TARGET )
                       ; and set Z flag correctly
```

0.7 CLRF

The instruction CLRF is not necessary because it can be replaced by a literal load 0 followed by a store.

```
;
;   Replacement for CLRF  TARGET
    MOVWF   TEMPW      ; save old W
    MOVLW   0
    MOVWF   TARGET     ; clear TARGET
    MOVF    TEMPW,W    ; restore W
    BSF     STATUS,Z   ; set Z flag correctly
;
```

0.8 MOVWF

The instrution MOVWF is not necessary because it can be replaced by two exclusive or's. This is tricky. Moreover, the Z flag must be preserved.

```
;
;   Replacement for MOVWF TARGET
   BCF     tempStatus,Z
   BTFSC   STATUS,Z
   BSF     tempStatus,Z   ; save Z flag
;
   XORWF   TARGET,W
   XORWF   TARGET,F   ; TARGET gets: ( W XOR TARGET ) XOR TARGET
                      ;    = W XOR ( TARGET XOR TARGET) = W XOR 0 = W
   MOVF    TARGET,W   ; restore W
;
   BCF     STATUS,Z
   BTFSC   tempStatus,Z
   BSF     STATUS,Z       ; restore Z flag
;
```

0.9 ADDWF

Addition can be replaced by subtraction — one first calculates the negative of one of the arguments. This can be done using increment and complement. Likewise subtraction can be replaced by addition. However, both can be replaced with a longer code sequence. Two solutions are given.

Replace ADDWF using logical operators. Addition can be done using exclusive or and logical and to to calculate in one bit column the sum and carry, and then carrying the carry to the next column by shifting the carry left one bit position and adding again. Stop when there are no further carries.

```
;    REPLACEMENT FOR ADDWF TARGET,d
;
;    Preamble:
;
     MOVWF  origW         ; w->origW
     MOVWF  tempA         ; w->tempA
     MOVF   TARGET,W      ; TARGET->tempB
     MOVWF  tempB
     MOVLW  0             ; 0->allCarries
     MOVWF  allCarries
 ;
 ;   Loop Invariant: add tempA and tempB,
 ;
 Loop
     MOVF   tempB,W
     XORWF  tempA,W
     MOVWF  tempW          ; tempB XOR tempA -> tempW
     MOVF   tempB,W
     ANDWF  tempA,W
     MOVWF  tempB          ; tempB AND tempA -> tempB
     IORWF  allCarries,F   ; tempB OR allCarries -> allCarries
     MOVF   tempW,W
     MOVWF  tempA          ; finish moving tempB XOR tempA into tempA
 ;
 ;   tempA has the 1-bit addition of tempA and tempB
 ;   tempB has the 1-bit carries, shift tempB and add again, if not zero
     BCF    STATUS,Z
     RLF    tempB          ; "carry" carries to the next column
     MOVF   tempB,F        ; set Z flag
     BTFSS  STATUS,Z
 ;   Assert Loop Invariant - tempA and tempB are to be added (again!,
 ;   because there were carries.)
     GOTO   Loop
 ;
 ;   Clean up
 ;
```

```
    BCF   STATUS,C
    BTFSC allCarries,7
    BSF   STATUS,C        ; set C flag correctly
    BCF   STATUS,DC
    BTFSC allCarries,3
    BSF   STATUS,DC       ; set DC flag correctly
;
    MOVF  tempA,W         ; result in W
    IORLW 0               ; set Z flag correctly
;   If d=0, code sequence ends here.
;   Else (d=1) continue with the following code sequence ...
    MOVWF TARGET          ; result in TARGET
    MOVF  origW,W         ; restore W
    MOVF  TARGET,F        ; set Z flag correctly
;
;   END OF ADDWF TARGET,d
;
```

0.10 ADDWF, version two

Replace ADDWF using increment/decrement. Addition can be done by counting up one term while counting done on the other. Carries are handled as they occur for a singe increment step.

```
;    REPLACEMENT FOR ADDWF TARGET,d
;
;   Preamble:

    MOVWF  origW        ; w->origW
    MOVWF  tempA        ; w->tempA
    MOVF   TARGET,W     ; TARGET->tempB
    MOVWF  tempB
    BCF    STATUS,C
    BCF    STATUS,DC
;
;   make sure tempB is not zero
    MOVF    tempB,F
    BTFSC   STATUS,Z
    GOTO    CleanUp
;
;   Loop Invariant: add tempB and tempA, tempB is not zero.
;
Loop
    INCF    tempA,F
```

```
;   check for carry and decimal-carry
    MOVF    tempA,W
    BTFSC   STATUS,Z
    BSF     STATUS,C  ; overflow, set carry
    ANDLW   0F
    BTFSC   STATUS,Z
    BSF     STATUS,DC ; low nibble overflow, set decimal carry
;
;   decrement tempB and loop if not zero
    DECFSZ  tempB,F
    GOTO    Loop
;
CleanUp
;
    MOVF  tempA,W        ; result in W
    IORLW 0              ; set Z flag correctly
;   If d=0, code sequence ends here.
;   Else (d=1) continue with the following code sequence ...
    MOVWF TARGET         ; result in TARGET
    MOVF  origW,W        ; restore W
    MOVF  TARGET,F       ; set Z flag correctly
;
;   END OF ADDWF TARGET,d
;
```

0.11 The Minimal Instruction Set

Consider these 11 instructions:

- ANDWF
- COMF
- MOVWF — because it affects no flags
- DECFSZ — some form of conditional branch is necessary.
- RLF — so that bits can affect each other “sideways”.
- CALL — necessary to push the stack.
- GOTO
- RETLW — replaces all literal commands.
- The special commands: SLEEP, CLRWDT, RETFIE.

The two questions are: can any instruction be simulated using only instructions from this list; and is there a list of less than 11 instructions for which this is also true?

I have verified that the answer to the first of the two questions is yes (try doing a BTFSS using a DECFSZ!), but I leave it to the student to show a shorter list or show that no shorter list is possible.