MSP430 Assembly Programlama vs C: Müfredat Örnekleri

Örnek 1: Bit Manipülasyonu ile Kaydırma

Problem: Bir kaydın bitlerini, belirli bir bit ayarlanana kadar sola döngü içinde bir pozisyon kaydırın.

Assembly Çözümü:
```assembly
MOV.W   #0x01, R4       ; R4'e başlangıç değerini yükle (0000 0001b)
MOV.W   #8, R5          ; 8 kaydırma için sayaç ayarla
LOOP:
   RLA.W   R4           ; Taşıma ile sola kaydır
   JNC     LOOP         ; Taşıma yoksa döngüyü tekrar et (bit ayarlı değil)
   ; Taşıma oluştuğunda döngüyü bitir
```

C Çözümü:
```c
unsigned int r4 = 1;
for (int i = 0; i < 8; i++) {
   r4 = r4 << 1;
   if (r4 & 0x100) break; // Taşımayı kontrol et
}
```

- **Assembly Neden Daha İyi**:
  - Donanım talimatlarının ("RLA" gibi) doğrudan kontrolü.
  - Fonksiyon çağrısı veya ekstra talimatlar yok.
  - Taşıma bayrağını verimli kullanma.

Örnek 2: Verimli Koşullu Atlamalar

Problem: Bir sayının tek mi çift mi olduğunu minimal talimatlarla belirleyin.

Assembly Çözümü:
```assembly
MOV.B   R4, R5          ; Kontrol edilecek sayıyı kopyala
AND.B   #0x01, R5       ; En az anlamlı bit ile maskele
JNZ     ODD             ; Sonuç sıfır değilse (tek) atla
EVEN:
   ; Çift durum için kod
   JMP     END
ODD:
   ; Tek durum için kod
END:
```

C Çözümü:
```c
if (number & 0x01) {
    // Tek durum için kod
} else {
    // Çift durum için kod
}
```

- **Assembly Neden Daha İyi**:
  - Assembly doğrudan işlemcinin koşullu dallanma talimatlarını kullanır ("JNZ" gibi) ve C'deki derleyici kaynaklı dallanma kodu ek yük oluşturmaz.
  - Ekstra değişken veya karşılaştırma işlemleri yoktur.

Örnek 3: Durum Makinesi Kullanımı

Problem: Durumları etiketlerle temsil edilen ve koşullara göre geçiş yapan basit bir durum makinesi uygulayın.

Assembly Çözümü:
```assembly
START:
    CMP     #1, R4       ; Mevcut durumu karşılaştır
    JEQ     STATE1       ; R4 == 1 ise STATE1'e atla
    CMP     #2, R4
    JEQ     STATE2       ; R4 == 2 ise STATE2'ye atla
    JMP     END          ; Varsayılan durum

STATE1:
    ; STATE1 için kod
    JMP     END

STATE2:
    ; STATE2 için kod
    JMP     END

END:
    ; Programın sonu
```

C Çözümü:
```c
switch (state) {
    case 1:
        // Durum 1 için kod
        break;
```

```
    case 2:
        // Durum 2 için kod
        break;
    default:
        // Varsayılan durum
        break;
}
```

- **Assembly Neden Daha İyi**:
  - Derleyicinin "atlama tablosu" veya karşılaştırma mantığı olmadan doğrudan dallanmaları uygular.
  - "Switch" ifadesinin veya fonksiyon çağrılarının ek yükünü barındırmaz.

Örnek 4: Yüksek Hızlı Sayaçlar

Problem: Bir kayıtta ayarlı bitlerin sayısını sayın.

Assembly Çözümü:
```assembly
MOV.W   R4, R5          ; Kayıt değerini kopyala
CLR.W   R6              ; Sayaçı temizle
LOOP:
    BIT.W   #1, R5      ; En az anlamlı biti test et
    JZ      SKIP        ; Bit sıfırsa artırmayı atla
    INC.W   R6          ; Sayaçı artır
SKIP:
    RRA.W   R5          ; Sağa kaydır
    JNZ     LOOP        ; Tüm bitler kontrol edilene kadar devam et
```

C Çözümü:
```c
unsigned int count = 0;
while (number) {
    if (number & 1) count++;
    number >>= 1;
}
```

- **Assembly Neden Daha İyi**:
  - Bit testi ("BIT.W") ve kaydırma ("RRA.W"), C'deki kayma ve maskeleme işlemlerinden daha verimlidir.
  - Derleyici, bu tür işlemler için daha fazla kod oluşturur.

Bu Örnekler Neden İyi Çalışıyor

1. **Koşullu Atlamalar**: Program akışının doğrudan kontrolünü gösterir.
2. **Kaydırma Komutları**: Mantıksal işlemler (AND veya OR gibi) kullanmadan bitlerin basit manipülasyonunu vurgular.
3. **Donanım Seviyesi Verimliliği**: Assembly, derleyicinin getirdiği soyutlama katmanını ortadan kaldırarak hassas optimizasyonlara izin verir.

Bu örnekleri göstererek, MSP430 işlemcilerde performans kritik görevler için assembly programlamanın avantajlarını etkili bir şekilde vurgulayabilirsiniz.