# MSP430 Tasarım Raporu

## 1. Giriş

Bu projede, Texas Instruments’ın düşük güç tüketimi ve yüksek verimlilik sunan MSP430 mikrodenetleyicisine yönelik bir assembler ve görsel arayüz geliştirilmiştir. Proje iki ana bileşenden oluşmaktadır:

* Assembler Modülü (assembler.py): MSP430 assembly dilinde yazılmış kodları derleyip makine koduna çeviren bir araç.
* Grafik Arayüz (gui.py): Kullanıcıların assembly kodlarını yazıp derleyebileceği, derleme sonuçlarını (makine kodu ve hata mesajları) görsel olarak takip edebileceği PyQt5 tabanlı bir uygulama.

## 2. MSP430 Hakkında Genel Bilgiler

MSP430, özellikle düşük güç tüketimi gerektiren uygulamalar için tasarlanmış 16-bit bir mikrodenetleyici ailesidir. Projede yer alan kodlar, bu mikrodenetleyicinin tipik özelliklerini ve assembly dilinin temel yapı taşlarını dikkate alacak şekilde geliştirilmiştir:

* Mimari Özellikler:
  + 16-bit işlemci yapısı.
  + Düşük güç tüketimi ve hızlı uyanma modu.
  + Zengin adresleme modları (register, indexed, symbolic, absolute, indirect, indirect with auto-increment, immediate).
* Komut Seti:
  + Format I: İki operandlı komutlar (ör. MOV, ADD, SUB, CMP, AND, XOR, vb.). Bu komutlar, kaynak ve hedef operandlara göre ek bellek kelimeleri gerektirebilir.
  + Format II ve III: Tek operandlı veya kontrol transferi komutları (ör. RRC, SWPB, CALL, JMP, JNE, JEQ, vb.).
  + Pseudo-ops: START, END, .DATA, .CODE, .ORG gibi sembolik direktifler, kodun başlangıç adresi, veri bölgesi veya kod bölgesi gibi düzenlemeleri yapmayı sağlar.
* Kullanılan Kavramlar:
  + Sembol Tablosu (SYMTAB): Assembly kodunda kullanılan etiketlerin adreslerini saklamak için.
  + Optab: Komut ve direktiflere karşılık gelen opcode değerlerini tutan sabit bir tablo.
  + Adresleme Modları: Assembly operandlarının türüne göre (örneğin, immediate veya register) farklı şekilde ele alınması.

Bu bilgiler doğrultusunda kod, MSP430’un temel çalışma prensiplerini ve mimari özelliklerini yansıtacak biçimde yapılandırılmıştır.

## 3. Kod Yapısı ve Yazım Süreçleri

### 3.1 Assembler Modülü (assembler.py)

Assembler, iki geçişli (two-pass) bir derleyici yaklaşımı kullanır:

* Pass1 (İlk Geçiş):
  + Assembly kod satır satır okunur.
  + Etiketler (labels) tespit edilerek sembol tablosuna (SYMTAB) eklenir.
  + Her komutun bellek adresi (LOCCTR) hesaplanır.
  + Pseudo-ops ve bilinen komutlar için adres artışları belirlenir.
  + Hatalar (örneğin, çift tanımlı etiketler veya bilinmeyen komutlar) kaydedilir.
* Pass2 (İkinci Geçiş):
  + Oluşturulan sembol tablosu kullanılarak gerçek makine kodu üretilir.
  + Format I komutları için, kaynak ve hedef operandların adresleme modları hesaplanır ve opcode bit alanları (örneğin, S-Reg, D-Reg, As, Ad) uygun şekilde yerleştirilir.
  + Ek operand kelimeleri gerekiyorsa (immediate veya absolute değerler) ilave bellek kelimeleri eklenir.
  + Format II/III komutları ve kontrol transferi (jump) komutları için özel hesaplamalar yapılır (örneğin, hedef adres offset hesaplaması).
  + Üretilen makine kodu, "output.hex" dosyasına yazılır.

Bu iki geçişli yapı, derleme sürecinde sembol çözümlemesi ve hata kontrolü için yaygın olarak tercih edilen bir yöntemdir.

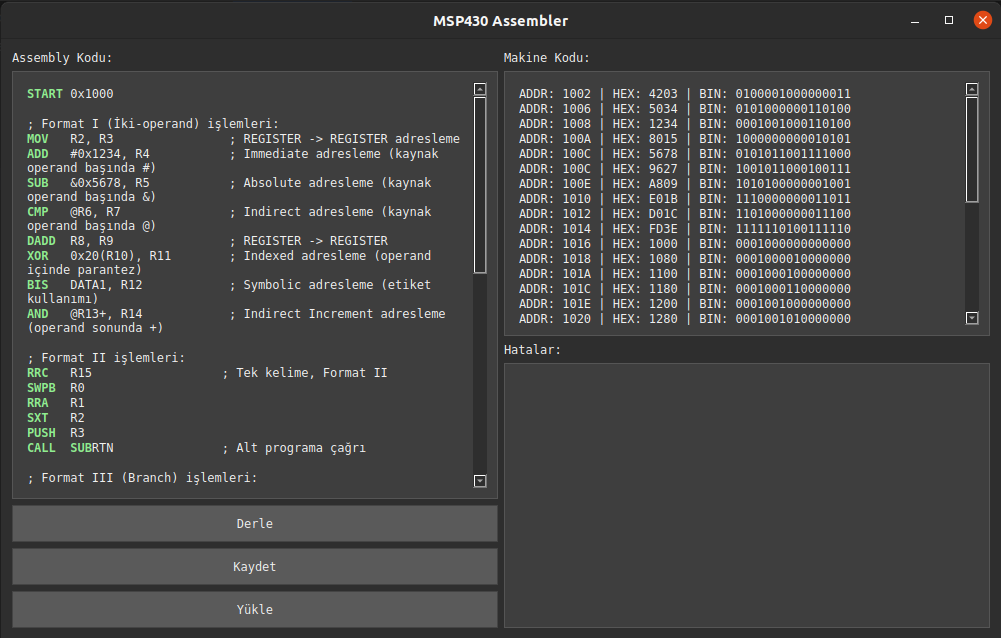
### 3.2 Grafik Arayüz (gui.py)

GUI, PyQt5 kütüphanesi kullanılarak geliştirilmiştir. Temel özellikler ve bileşenler şunlardır:

* Kod Editörü:
  + Kullanıcının assembly kodunu girebileceği QPlainTextEdit alanı.
  + Özel bir syntax highlighter (Highlighter sınıfı) ile assembly diline özgü anahtar kelimeler vurgulanır.
* Çıktı Ekranları:
  + Derlenen makine kodunu gösteren bir QTextEdit.
  + Oluşan hata mesajlarını listeleyen ayrı bir QTextEdit.
* Fonksiyonellik:
  + Derle (Assemble) Butonu: Kullanıcının yazdığı kodu derler, makine kodu ve hata mesajlarını günceller.
  + Kaydet / Yükle Butonları: Assembly kodunu dosyaya kaydetme ve daha önce kaydedilmiş kodu yükleme işlevselliği sağlar.
* Dinamik Modül İçe Aktarma:
  + GUI, dinamik olarak assembler modülünü yükler (importlib ile), böylece assembler kodundaki güncellemeler arayüze yansıtılabilir.

## 4. Arayüz Tasarımı ve Örnek Çalışma Kodları

### 4.1 Arayüz Tasarımı



## 5. Kod Yazım Süreçleri ve Geliştirme Aşamaları

Projenin geliştirilme aşamaları şu şekilde özetlenebilir:

* Analiz ve Tasarım:
  + MSP430 mimarisi ve komut seti detaylı olarak incelendi.
  + İki geçişli derleyici (pass1 ve pass2) mantığı belirlendi.
  + Hata yönetimi ve sembol tablosu yapısı oluşturuldu.
* Kodlama:
  + Assembler Modülü: Sembol tablosu, opcode tablosu, adresleme modları ve iki geçişli derleyici adımları (pass1 ve pass2) dikkatlice kodlandı. Kod yazım sürecinde, MSP430’un gerçek donanım mimarisine uygunluk göz önünde bulunduruldu.
  + Grafik Arayüz: PyQt5 kullanılarak modern bir kullanıcı arayüzü geliştirildi. Kod editöründe syntax highlighting, dosya yükleme/kaydetme ve derleme sonrası sonuç gösterme özellikleri eklendi.
* Test ve Hata Ayıklama:
  + Her iki modül entegre edilerek kullanıcı testleri gerçekleştirildi.
  + Derleme sırasında oluşan hata mesajları, kullanıcı dostu bir formatta sunuldu.
* Dokümantasyon:
  + Kod içerisindeki yorumlar ve modüler yapı, ileride yapılacak eklemeler için temel oluşturdu.

## 6. Gelecekte Eklenmesi Planlanan Özellikler

Projenin mevcut hali temel bir assembler ve grafik arayüz sunarken, ilerleyen aşamalarda eklenmesi planlanan bazı geliştirmeler şunlardır:

* Genişletilmiş Komut Seti Desteği:
  + MSP430’a özgü daha fazla komut ve pseudo-op eklenmesi.
  + Kullanıcının daha kapsamlı kod yazabilmesi için ek makro tanımları.
* Gelişmiş Hata Ayıklama Özellikleri:
  + Gerçek zamanlı hata vurgulama.
  + Hata mesajlarına yönelik detaylı açıklamalar.

## 7. Sonuç

Bu proje, MSP430 mikrodenetleyicisinin assembly diline yönelik geliştirilmiş bir derleyici (assembler) ve kullanıcı dostu bir grafik arayüzün entegre edildiği kapsamlı bir çalışma olarak ortaya çıkmıştır. Hem kod yapısı hem de arayüz tasarımı, gelecekteki genişletmeler ve optimizasyonlar için esnek bir altyapı sunmaktadır. İlerleyen süreçlerde eklenmesi planlanan özellikler, kullanıcı deneyimini ve derleyici performansını daha da iyileştirmeyi hedeflemektedir.