Tam Bypasslı Pipeline Modelin Tamamlanması

ve

3 Yeni Komut Eklenmesi

Rapor Özeti / İçindekiler

1) Bu programı neden seçtim/ uyguladım ?

Bu projeyi seçmemin temel nedeni, ileride donanım tasarımı ve işlemci mimarisi alanında çalışmayı hedeflememdir. En büyük hedeflerimden biri, kendi işlemcimi tasarlayabilmektir. Bu amaç doğrultusunda, RISC-V tabanlı bir işlemci mimarisini adım adım inşa etmek, bana hem teorik hem de pratik açıdan önemli bir deneyim kazandırdı.

Daha önce RISC-V alanında nasıl bir başlangıç yapmam gerektiğini bilmiyordum. Bu proje sayesinde, alana dair gerekli teorik bilgileri, simülasyon uygulamalarını ve çeşitli pratik becerileri edindim.

---

2) Projede neden bu alanı seçtim?

Bir önceki maddede belirttiğim gibi, ileride kendi işlemci tasarımımı oluşturmak istiyorum. Bu hedefim doğrultusunda, mikromimari üzerinde yoğunlaşmak ve optimize bir tasarım ortaya çıkarmak için en uygun alanın bu proje olduğunu düşündüm.

Projenin RISC-V tabanlı bir işlemci mimarisi etrafında şekillenmesi, hem mimariyi derinlemesine anlamamı sağladı hem de donanım tasarımı konusundaki becerilerimi geliştirmeme yardımcı oldu.

Bu nedenle, projede işlemci mimarisi ve RISC-V entegrasyonu üzerine çalışmayı özellikle tercih ettim.

---

3) Tasarım başlangıcı ve benim katkım

Derslerde birlikte ilerlediğimiz projenin tek parça ve bütünleşik bir yapıda olması, mimariyi tam olarak kavramamı zorlaştırdı. Bu nedenle, işe sıfırdan ve tek çevrimlik bir işlemci tasarlayarak başladım. Sonrasında ise bu mimariyi aşama aşama beş kademeli bir boru hattı (pipeline) yapısına dönüştürmeye çalıştım; ancak süreçte pek çok hatayla karşılaştım ve bilgi eksikliğim nedeniyle bazı problemleri çözmekte zorlandım.

Yaptığım projenin sürdürülebilir olmadığını fark edince, derste uygulama olarak üzerinde çalıştığımız ve henüz tamamlanmamış olan projeye yönelmeye karar verdim. Öncelikle projenin genel yapısını, sinyallerin görevlerini ve modüller arasındaki ilişkileri detaylı bir şekilde analiz ettim. Bu analizlerin ardından, projeyi boru hattı mimarisine uyarlamaya ve gerekli işlevleri adım adım eklemeye başladım.

Çalışmama, öncelikle boru hattı yapısındaki temel sorunları çözerek başladım. Veri bağımlılıkları (data hazards), kontrol bağımlılıkları (control hazards), bekletme (stall), flush (talimat temizleme), yönlendirme (forwarding) gibi boru hattına özgü çeşitli problemleri tespit edip çözümler geliştirdim. Ardından projede istenen üç adet bit manipülasyon komutunu (clz, ctz, cpop) mimarime entegre ettim. Son olarak ise, verilen testlerin yetersiz olduğunu göz önünde bulundurarak, kendi geliştirdiğim ek testleri projeye dahil ettim.

Bu temel yapının üzerine eklenen yeni özellikler ve iyileştirmeler ise şöyledir:

- **Boru Hattı Mimarisi (Pipeline):** Yarım kalan projede mevcut olan eksiklikleri analiz ettikten sonra, işlemci mimarisini beş aşamalı bir boru hattı yapısına dönüştürüldü. Her bir aşamayı modüler olarak tasarlanarak, sistemin esnekliği ve anlaşılabilirliği artırıldı. Ardından bu modülleri merkezi bir çekirdek model (core model) içerisinde bir araya getirilerek, boru hattı mimarisinin tüm işlevleri entegre edildi.

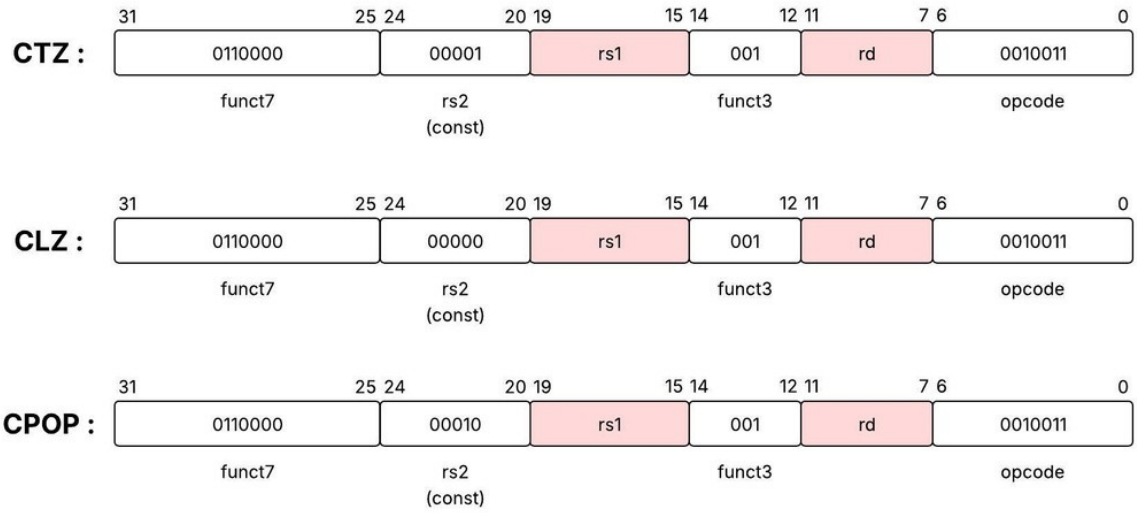
core_model.sv	Tüm mimarinin birleştiği ana modüldür. Boru hattı yapısı ve tehlikelerinin (hazard) çözümleri burada entegre edilir.
fetch.sv	Komutun bellekten alınmasını (fetch) ve PC güncellenmesini sağlar.
decode.sv	Komutun çözümlenmesi, yapılacak işlemlerin sinyallerin oluşturulmasını sağlar.
execute.sv	ALU işlemleri ve dallanma (branch/jump) kararlarının alındığı aşamadır.
memory.sv	Bellek okuma ve yazma işlemlerinin gerçekleştirildiği modüldür.

write_back.sv	Hesaplanan sonucun register dosyasına geri yazıldığı (write-back) aşamadır.

Tablo 1.0 : *Projenin modüler yapısının açıklanması*

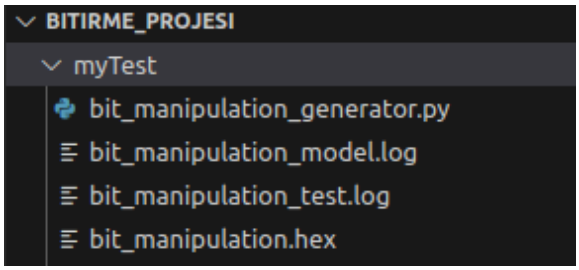
- **Boru Hattı Problemlerinin Çözülmesi:** Oluşturulan boru hattı mimarisi, doğası gereği bazı belirsizlikler ve veri/akış bağımlılıkları içerdiği için, aşağıdaki temel sorunlara yönelik çözümler geliştirildi:
    - **Veri Yönlendirme (Forwarding):** Boru hattında ardışık komutlar arasında veri bağımlılığı oluştuğunda, gerekli veriler ileriye taşınarak beklemelerin önüne geçildi.
    - **Bekletme (Stall):** Forwarding'in yeterli olmadığı durumlarda, ilgili komutun yürütülmesi geciktirilerek doğru veri kullanılana kadar boru hattı geçici olarak durduruldu.
    - **Boru Hattı Temizleme (Flush):** Dallanma (branch) ve atlama (jump) komutlarından sonra, yanlış yürütülen veya gereksiz aşamalardaki komutlar temizlenerek doğru akış sağlandı.
    - **Veri Tehlikeleri (Data Hazards):** Kaynak ve hedef register'lar arasında oluşan veri bağımlılıkları analiz edilip, uygun önlemler alındı.
    - **Kontrol Tehlikeleri (Control Hazards):** Dallanma sonrası oluşabilecek belirsizlikler için, doğru komutun yürütülmesi garanti altına alındı.

- **3 Yeni Komutun Eklenmesi:** Projede istenen üç bit manipülasyon komutu (CLZ, CTZ, CPOP) mimariye entegre edildi

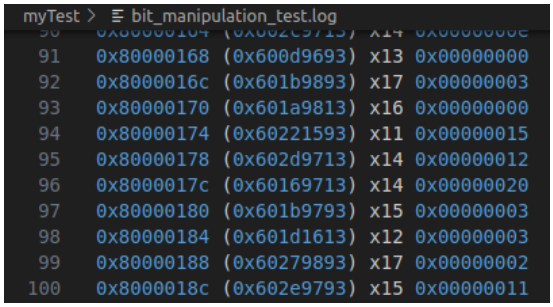


Görsel 1.0 : *Eklenen komutların buyruk yapısı.*

Bu komutlar ALU'da ek işlevler olarak tanımlandı ve doğru çalışıp çalışmadıklarını doğrulamak için, her buyruk sonrası log dosyası oluşturan bir Python programı geliştirildi. Oluşturulan log dosyası ile boru hatlı işlemcinin çıktısı karşılaştırılarak, eklenen komutların doğru çalıştığı teyit edildi.



Görsel 1.1 : *Proje yapısında komutların test edilmesini sağlayan dosyalar.*



Görsel 1.2 : *"bit_manipulation_test.log" dosyasının içeriği*

- **Test Kapsamının Genişletilmesi ve Ek Testlerin Oluşturulması :** Projede başlangıçta kullanılan "test.log" dosyasındaki testlerin, yalnızca işlemcinin temel işlevlerini kontrol ettiği ve özellikle boru hattı mimarisi için kapsamlı bir doğrulama sağlamadığı görüldü.

  Boru hattı yapısına özgü veri ve kontrol tehlikeleri, bekletme (stall), yönlendirme (forwarding) ve flush gibi önemli senaryoların yeterince test edilmediği tespit edildi. Bu eksikliği gidermek amacıyla, boru hattı mimarisinin karmaşık durumlarını ve işlemcinin hafıza birimini kapsamlı şekilde sınayan yeni testler geliştirildi. Eklenen bu testler sayesinde, işlemcinin farklı koşullarda ve özel durumlarda da doğru çalışıp çalışmadığı ayrıntılı olarak doğrulanmış oldu.

  Tüm testler, Docker container ortamında Spike simülatörü ile çalıştırıldı ve her testin çıktısı log dosyası olarak kaydedildi.

data_hazard_test.log	veri tehlikelerini test eden log.
jalr_test.log	“jump and link register” komutunun test eden log.
memory_test.log	hafıza birimini test eden log.
stall_test.log	bekletme durumunu hafıza işlemleri ile beraber test eden log.

Tablo 1.1 : *Eklenen testlerin açıklanması.*

4) Hazard, Stall ve Flush Yapılarının Tasarımı ve Gerekçeleri

Bu proje kapsamında, boru hattı mimarisinde karşılaşılan veri ve kontrol bağımlılıkları (hazard) çözmek amacıyla çeşitli mekanizmalar tasarlandı ve entegre edildi.

Veri bağımlılıklarını (data hazards) önlemek için öncelikle **forwarding (veri yönlendirme)** mekanizmasını uyguladım. Böylece, ihtiyaç duyulan veri daha sonraki aşamalardan alınarak boru hattındaki beklemeler en aza indirildi.

Forwarding'in yeterli olmadığı durumlarda ise **stall (bekletme)** mekanizmasını kullandım; bu sayede, veri hazır olana kadar ilgili komutun yürütülmesi geçici olarak durduruldu ve yanlış veri kullanımının önüne geçildi.

Ayrıca, dallanma (branch) komutları gibi kontrol tehlikesi (control hazard) oluşturan durumlarda **flush (boru hattını temizleme)** yapısını entegre ettim. Yanlış tahmin edilen veya alınmayan dallanmalardan sonra, hatalı komutların yürütülmesi engellenerek sistemin doğru akışta devam etmesi sağlandı.

Tüm bu optimizasyonlar sayesinde, boru hattı mimarisine özgü veri ve kontrol tehlikeleri etkili şekilde yönetildi ve işlemcinin güvenilirliği ile performansı artırıldı.

Çözüm Yöntemlerinin Seçilme Nedenleri

Alternatif olarak, pipeline'da bubble (nop eklemek) veya daha karmaşık dinamik tahminci (branch predictor), out-of-order execution gibi ileri düzey optimizasyonlar da uygulanabilirdi. Ancak:

- Bubble/NOP eklemek veri veya kontrol tehlikesi oluşan her durumda boru hattına boş komut (nop) eklemek anlamına gelir. Bu yöntem, basit olsa da, işlemcinin performansını ciddi şekilde düşürür; çünkü işlemci verimli şekilde çalışmak yerine gereksiz döngüler harcar.

- Daha gelişmiş yöntemler (ör. dinamik branch prediction, speculative execution, out-of-order execution) modern işlemcilerde yüksek performans sağlasa da; hem tasarımı hem de doğrulaması çok daha karmaşıktır ve bu proje kapsamının ötesindedir.

Bu projedeki öncelik; hem donanım tasarımının temel ilkelerini kavramak hem de boru hattı mimarisinin klasik veri ve kontrol tehlikelerine karşı etkili ve anlaşılır çözümler üretmektir. Bu nedenle, forwarding, stall ve flush gibi hem öğretici hem de pratikte sık kullanılan yöntemler tercih edildi.

5) Tasarım ve Test Sürecinde Karşılaşılan Sorunlar ve Çözüm Yöntemleri

1. **Control Hazards (Kontrol Tehlikeleri):** Branch ve jump komutlarında pipeline'ın hangi komutları getireceği belirsizliği

Çözümler:

a. **Pipeline Flush:** Yanlış tahmin durumunda pipeline'ı boşalt.
b. **Branch Prediction:** Statik (her zaman taken/not-taken) tahmin.

Projede Uygulanan Yapı:

Dallanma (branch) komutlarında, işlemci her zaman statik olarak "dallanma alınmayacak" varsayımıyla çalışır. Gerçek dallanmanın olup olmadığı ise execute aşamasında belirlenir. Eğer execute aşamasında bir dallanma gerçekleşirse, boru hattının decode ve execute aşamalarında yer alan ve artık geçersiz olan sinyaller ve değerler temizlenir (flush edilir). Bu sayede, yanlış yönde ilerlemiş komutların etkisi ortadan kaldırılır. Loglama aşamasında ise, bu temizlenen ve geçersizleşen sinyaller log dosyasına dahil edilmez.

2. **Data Hazards (Veri Tehlikeleri):**

RAW (Read After Write) Hazards:

i. **Forwarding/Bypassing:** Sonucu henüz register'a yazılmadan doğrudan kullanan komuta aktar.
ii. **Pipeline Stalling/Bubbling:** Gerekli veri hazır olana kadar pipeline'ı durdur.

Projede Uygulanan Yapı:

Projede, veri tehlikelerine (data hazards) karşı iki temel çözüm uygulandı:

Öncelikle, komutlar arasındaki RAW (Read After Write) tehlikelerini azaltmak için **forwarding (bypassing)** mekanizması kullanıldı. Böylece, bir komutun henüz register dosyasına yazılmamış sonucu, doğrudan ihtiyaç duyan sonraki komuta iletilerek boru hattındaki gereksiz beklemeler önlendi.

Ancak forwarding ile çözülemeyen durumlar (örneğin, ardışık load-use ilişkilerinde veri henüz hazır değilse), **pipeline stalling** yöntemiyle ilgili komutun yürütülmesi geçici olarak durduruldu. Bu sayede, gerekli veri hazır olana kadar boru hattında hatalı işlem yapılmasının önüne geçildi.

3. **Structural Hazards (Yapısal Tehlikeler):**

Çözümler:

a. **Pipeline Stalling:** Kaynak uygun olana kadar bekle.
b. **Resource Scheduling:** Kaynakları zaman dilimlerine böl.

Projede Uygulanan Yapı:

Projede yapısal tehlikeler ile karşılaşılmadı; çünkü her aşama için gerekli donanım kaynakları (register dosyası, ALU, bellek) ayrı ayrı tasarlandığından kaynak çakışması olmadı. Ancak, olası bir yapısal tehlike durumunda, önceki veri tehlikelerinde uygulanan **pipeline stall (bekletme)** mekanizması kullanılacak şekilde mimari hazırlandı.

6) Farklı Testlerde Ortaya Çıkan Hatalar ve Çözüm Yolları

Pipeline Register Senkronizasyonu Sorunu:

Problemler:

Komutun loglanmasında 4 cycle gecikme yaşandı, loglar golden model ile senkronize gitmedi.

Görsel 1.3 : *Senkronizasyon probleminin waveform'da gösterilmesi.*

Çözüm:

Her aşamanın bilgisini kendi pipeline register'ında ayrı olarak tutuldu ve loglama işlemini writeback aşamasında güncel bilgilerle gerçekleştirildi.

Flush ve Stall Mekanizmasında Yanlış Loglama:

Problemler:

Flush edilen veya geçersiz komutlar loglara gereksiz şekilde yansıyordu.

Çözüm:

Valid sinyalleri ve reset/flush kontrolleriyle sadece geçerli (işlenen) komutların loglanmasını sağlandı.

Ayrıca, testbench dosyası olan tb.sv içinde loglama işlemi yapılırken, her clock çevriminde komutun güncelliği ve geçerliliği update

sinyaliyle kontrol edildi. Ancak, doğrudan clock kenarında log alınmaya çalışıldığında, valid sinyali 0 olsa bile log dosyasına gereksiz satırlar yazılıyordu.
Bunu önlemek için, log satırlarının yazılmasında her döngüde #2 zaman gecikmesi (delay) eklendi. Böylece, clock ve loglama işlemleri arasında senkronizasyon sağlanmış oldu.

Data Hazard Sonucu Yanlış Sonuç Üretimi (Load-Use):

Problemler:

Ardışık load ve use komutlarında veri forwarding eksikliğinden kaynaklı yanlış veri yazıldı.

Çözüm:

Forwarding/bypassing mantığını hem rs1/rs2 hem de store (sw) için ekleyerek güncel veri ile işlem yapılmasını sağlandı, gerekli durumlarda stall eklendi.

Branch Komutlarından Sonra Hatalı Komut Çalışması:

Problemler:

Dallanma sonrası, boru hattında yanlış komutlar yürütüldü.

Çözüm:

Branch sonrası pipeline flush mekanizmasını düzgün şekilde devreye aldım, decode ve execute aşamalarındaki komutları temizledim.

Bellek (Memory) İşlemlerinde Adresleme Sorunu:

Problemler: Memory erişimlerinde adres kaymaları veya yanlış veri yazımı/okuma.

Çözüm: Adres hesaplamalarını ve memory erişim sinyallerini

dikkatlice kontrol ettim, testler ile doğruladım.

Genel olarak, sinyallerin doğru şekilde iletilmemesi ve boru hattı problemlerinin uygun sinyallerle çözülememesi gibi sorunlarla karşılaşıldı. Bu tür problemleri tespit etmek ve analiz etmek amacıyla çoğunlukla waveform (dalga formu) gözlemlerinden yararlanıldı.

7) Geri Bildirim ve Değerlendirme

RISC-V'in teorik anlatımını genel olarak oldukça başarılı buluyorum. Dersi anlatan hocamızın mimari konulardaki derin bilgisi ve pratiğe dayalı yaklaşımı, kavramların daha kalıcı ve anlaşılır olmasını sağladı. Uygulama aşamasında ise, bazı konularda teorik bilgi ile pratikte karşılaşılan detayların tam olarak örtüşmemesi, zaman zaman soyutlamaları anlamamı zorlaştırdı. Bu noktada, uygulama dersini veren hocamızın öğrenciye destek olan yaklaşımı ve her zaman yardım etmeye açık olması sayesinde karşılaştığım sorunlarda hızlıca çözüm bulabildim ve projeyi başarıyla tamamlayabildim.

Bu proje sürecinde yaşadığım sorunlar ve üstesinden gelme sürecim, alanımda kendimi geliştirmem için önemli bir fırsat sundu. Tüm katkıları için hocalarıma ve Milli Teknoloji Akademisi'ne teşekkür ediyorum.

Ad-Soyad: Furkan Yıldırım
E-posta: yldrmfurkan77@gmail.com
Dönem: 6. Dönem
Bölüm: Bilgisayar Mühendisliği