ÖNSÖZ

Bu rapor, RISC-V mimarisini temel alan RV32-I işlemcisinin VHDL ile yazılma sürecini detaylı bir şekilde incelemektedir. RISC-V, açık kaynaklı ve geniş topluluk desteğine sahip olmasıyla dikkat çeken bir ISA standardını temsil etmektedir. Bu rapor, bu açık standart mimariyi temel alan RV32-I işlemcisini, FPGA teknolojisi kullanarak fiziksel bir platforma taşıma çabalarını içermektedir.

Raporda, FPGA üzerinde donanım düzeyinde bir işlemci gerçekleme süreci adım adım ele alınacak ve her aşama detaylı bir şekilde incelenecektir. İşlemcinin tasarımı bu raporun odak noktasını oluşturacaktır.

Bu rapor, RISC-V mimarisi ve FPGA tabanlı işlemci tasarımı konularına ilgi duyan okuyuculara, açık standartlar ve donanım tasarımıyla ilgili deneyim kazanmak isteyen mühendis ve araştırmacılara yöneliktir.

İÇİNDEKİLER

SİMGELER VE KISALTMALAR LİSTESİ

ISA	: Instruction Set Arthitecture
FPGA	: Field-programmable gate array
VHDL	: VHSIC (Very High Speed Integrated Circuits Program) Hardware Description Language
RISC	: Reduced Instruction Set Computer
ASIC	: Application Specific Integrated Circuit
IDE	: Integrated Development Environment
KiB	: Kilo Binary Bit

İNGİLİZCE – TÜRKÇE SÖZLÜK

Pipeline	: Boru Hattı
Operand Forwarding	: Veri Yönlendirmesi
Instruction	: Buyruk
Instruction Set Arthitecture	: Buyruk Kümesi Mimarisi
Register	: Yazmaç
Register File	: Yazmaç Öbeği
Memory	: Bellek
Arithmetic Logic Unit	: Aritmetik Mantık Birimi
Program Counter	: Program Sayacı
Branch	: Dallanma
Branch Prediction	: Dallanma Öngörüsü
Immediate	: Anlık
Base Instructions	: Temel Buyruklar
Extension Instructions	: Ek Buyruklar
Control Hazard	: Denetim Sorunu
Data Hazard	: Veri Sorunu
Process	: İşlem
Fetch	: Getir
Decode	: Çöz
Execute	: Yürüt
Write Back	: Geri Yaz
Data Path	: Veri Yolu
Hardware Description Language	: Donanım Tanımlama Dili
Application Specific Integrated Circuit	: Uygulamaya Özel Tümleşik Devre
Program Counter	: Program Sayacı
Control Unit	: Denetim Birimi
Immediate Generator	: Anlık Üreteci
Assembly	: Çevirici Dil
Stall	: Durdurma
2-bit Saturating Counter Predictor	: Çift Doruklu Öngörücü

ŞEKİLLER LİSTESİ

ÖZET

Anahtar kelimeler: RISC, RV32-I, Dallanma Öngörüsü, Veri Yönlendirmesi, Boru Hattı, VHDL, FPGA

Bu çalışmada önceki tasarım projesinde yapılan işlemcideki birçok problem ve eskiklik giderilmiştir. Önceki işlemcideki en büyük problemlerden biri olan birimlerin birbirleriyle entegre edilmemesi problemi bu çalışmada giderilmiştir. İşlemcideki dallanma öngörücüsü sistemi ve veri yönlendirme mekanizmaları uygun bir şekilde oluşturulmuş ve diğer birimlere entegre edilmiştir. Ayrıca diğer bütün parçalar da birbirleriyle bağlanmıştır.

Önceki işlemcideki entegre problemlerini gidermek için sistemdeki bazı birimler değiştirilmiş, çıkarılmış veya sisteme yeni birimler eklenmiştir. Birimlerdeki benzer değişiklikler buyruklarda da olmuştur. İşlemciye kriptografi işlemlerini daha hızlı yapabilmesi için 2022 senesindeki Teknofest Çip Tasarım yarışmasında [1] ister olarak belirtilen özel buyruklar eklenmiştir. Böylece işlemci, kriptografi işlemleri için donanımsal hızlandırma bulundurarak diğer işlemcilere kıyasla büyük avantaj sağlaybilecek hale gelmiştir.

Geliştirilen işlemci simülasyon üzerinde gayet düzgün çalışmasına rağmen FPGA gerçeklemesinde maalesef ki problemler vardır.

BÖLÜM 1. GİRİŞ

Tam sayılar üzerinde temel işlemler yapan ve kriptografi işlemlerini hızlandırıan bir işlemci geliştirilmiştir.

1.1. Temel Özellikler

Tasarlanan işlemci 32 bitlik açık kaynak RISC-V buyruk kümesi mimarisine göre, bellek sistemi ise Hardvard mimarisine göre Little Endian olarak tasarlanmıştır. Temel RV-32I buyruklarının yanında kriptografi işlemlerini hızlandıran özel buyrukları destekler. Klasik RISC boru hattı olan Getir, Çöz, Yürüt, Bellek ve Geri Yaz olarak adlandırabileceğimiz 5 farklı aşamaya sahip bir boru hattı vardır. Bu boru hattındaki kontrol sorunlarını çözmek için dallanma öngörücüsü kullanılmıştır. Veri sorunları içinse veri yönlendirmesi ve durma yöntemleri kullanılmıştır.

1.2. Amaç

Bu işlemciyi geliştirmekteki genel amaç işlemci geliştirme konusunda ar-ge yapmak ve bu alandaki kaynak sayısını arttırmaktır. Her geçen gün işlemci geliştirmenin önenimini daha fazla arttığı bu dönemlerde bu alanda uğraşmak isteyenler için faydalı olacağı düşünülmüştür.

1.3. Maliyet

İşlemcinin geliştirilmesinde FPGA dışında hiç bir maliyet olmamıştır. Kullanılan bütün yazılımlar ve araçlar ücretsizdir. FPGA olarak okul tarafından sağlanan Altera DE2-115 kullanılmıştır. Değeri $423'dır [2].

1.4. İş Çizelgesi

1 – 4 → İşlemci birimlerinin yeniden tasarlanması
5 – 8 → Dallanma öngörüsü gerçeklenmesi
9-12 → Veri yönlendirmesinin gerçeklenmesi
9 – 14 → Sistemin doğrulanması

BÖLÜM 2. KULLANILACAK TEKNOLOJİLER

Bu projede, bir teknoloji güçler erkini bir araya getirerek öne çıkıyor: RISC-V mimarisi, VHDL donanım tanımlama dili ve FPGA teknolojisi. Projenin farklı aşamalarında önemli roller üstlenen bu teknolojiler, beraber çalışarak donanım tabanlı bir RV32-I işlemcisi geliştirilmesine yardım sağlıyor.

2.1. RISC-V Buyruk Kümesi Mimarisi

Şekil 2.1 RISC-V'in Logosu [3]

RISC-V, açık kaynaklı bir RISC tabanlı buyruk kümesi mimarisidir. Prof. Krste Asanović, ve yüksek lisans öğrencileri Yunsup Lee ve Andrew Waterman, RISC-V buyruk setini 2010 Mayıs'ında UC Berkeley'deki Paralel Hesaplama Laboratuvarı'nda (Par Lab) Prof. David Patterson'ın direktörlüğünde başlattılar. Birçok RISC-V işlemcisini tasarlamak için kullanılan Chisel donanım tasarlama dili de Par Lab'de geliştirilmiştir [4]. RISC-V, modüler tasarımı sayesinde farklı uygulamalara uygun olarak özelleştirilebilir.

RISC-V bunu temel buyruklarının yanında ek buyruklar da bulundurarak yapar. Geliştirici projenin amacına uygun olan ek buyrukları ekleyerek işlemcisini geliştirebilir. Bu açık standart, düşük maliyetli ve enerji verimli çözümler sağlar, aynı zamanda büyük ve küçük ölçekli sistemlerde kullanılabilir.

2.1.1. Neden RISC-V

RISC-V'in cazibesini, açık kaynak doğası, ücretsiz erişim imkânı ve bir dizi çeşitli projede başarıyla kullanılabilmesi oluşturuyor. Bu mimari, kullanıcılarına geniş bir

özelleştirme ve adaptasyon alanı sunarak, farklı uygulama alanlarında esnek ve maliyet etkin çözümler geliştirmelerine olanak tanır.

Ayrıca, RISC-V'in benimsenmesi, geniş bir topluluğun desteği ve sürekli büyüyen ekosistemi ile, inovasyona ve gelişime katkı sağlamaktadır. Bu nedenlerle, RISC-V tercih etme kararı hem teknik hem de ekonomik açıdan avantajlı bir seçenektir.

2.1.2. Maliyet

Bu mimarinin ücretsiz oluşu, bir dizi avantajı beraberinde getirir ve bu avantajlar onu, hem endüstri profesyonelleri hem de akademisyenler için çekici kılar. RISC-V, kullanıcıların mimariyi ihtiyaçlarına göre özelleştirmesine ve geliştirmesine olanak tanır. Bu, endüstrideki birçok uygulama ve projede esneklik sağlar.

RISC-V Vakfı tarafından belirlenen ücretsiz ve açık kaynaklı lisanslama modeli, mimarinin ücretsiz kullanılmasını sağlar. Bu, RISC-V tabanlı işlemcilerin ticari veya akademik projelerde ücretsiz olarak kullanılabilmesine olanak tanır. Geleneksel ISA mimarilerinin aksine, RISC-V'nin kullanımı için herhangi bir ücret ödenmesine gerek yoktur.

RISC-V, dünya genelinde geniş bir topluluk tarafından desteklenmektedir. Bu topluluk, sürekli olarak mimariyi geliştirir ve yeni özellikler ekler. Topluluk üyeleri, RISC-V'nin sürekli olarak evrimleşmesine katkıda bulunarak, kullanıcıların güncel ve gelişmiş bir mimariye erişimini sağlar.

Ücretsiz ve açık kaynaklı doğasıyla RISC-V, eğitim kurumları, araştırma laboratuvarları ve öğrenciler için ideal bir seçenek sunar. RISC-V mimarisinin bu yapısı, endüstri içinde inovasyonu teşvik eder.

Küçük şirketler ve girişimciler, düşük maliyetle kendi özel işlemcilerini tasarlayabilir ve üretebilir. Bu durum, geniş bir uygulama yelpazesi için özel işlemci tasarımlarının ortaya çıkmasına olanak tanır.

2.1.3. Gerçeklenecek Olan Buyruklar

Projede RISC-V'in 32 bit temel buyruk kümesi olan RV32-I gerçeklenecektir. Bu, basitleştirilmiş buyruk kümesi kullanılarak düşük güç tüketimi ve yüksek verimlilik hedeflenirr.

Bu temel buyruk kümesi, yüksek performanslı gömülü sistemler, mikro denetleyiciler ve diğer düşük güçlü uygulamalarda kullanım için seçilmiştir. RV32I, RISC-V'nin genişletilebilir yapısının temelini oluşturur ve daha karmaşık buyruk kümeleri veya özelliklerle genişletilebilir.

2.2. VHDL

VHDL, bir donanım tanımlama dilidir. VHDL ilk olarak 1983 yılında Amerika Birleşik Devletleri Savunma Bakanlığı'nın emriyle, tedarikçi firmaların ekipmanlarına dahil ettikleri ASIC'lerin davranışlarını belgelemek amacıyla geliştirilmiştir.

VHDL, elektronik sistemlerin tüm aşamalarında kullanılmak üzere tasarlanmış resmi bir notasyondur. Hem makine tarafından okunabilir hem de insan tarafından okunabilir olması nedeniyle, donanım tasarımlarının geliştirilmesini, doğrulanmasını, sentezlenmesini ve test edilmesini; donanım tasarım verilerinin iletilmesini; ve donanımın bakımını, değiştirilmesini ve teminini destekler. Başlıca hedef kitlesi, dili destekleyen araçların uygulayıcıları ve dilin ileri düzeyde kullanıcılarıdır [5].

Dilin adındaki VHSIC kısaltması, standart üzerindeki erken çalışmaları finanse eden ABD hükümeti programından gelmektedir [6].

2.2.1. Nerelerde Kullanılır

Elektronik devre tasarımında, özellikle FPGA ve ASIC gibi dijital tasarım platformlarında yaygın olarak kullanılır. VHDL ile özel amaçlara hizmet edecek birçok donanım geliştirilebilir. Örneğin bir kablodan gelen veriyi farklı bir protokole göre işleyen bir kodlayıcının tasarlanmasında, etraftan alınan görüntüleri işleyen bir

sistemde görüntü işleme algoritmalarını donanım tarafında gerçeklenmesini veya yapay zekâ hızlandırıcı bir donanım geliştirilmesinde kullanılabilir.

Ayrıca mikro denetleyici ve diğer gömülü sistemlerin tasarımında da kullanılır. Bu, mikro denetleyici tabanlı uygulamaların geliştirilmesini ve optimize edilmesini sağlar.

2.2.2. Maliyet

VHDL, genel olarak bir ücretli lisanslama sistemine tabi değildir ve geniş bir kullanıcı kitlesine ücretsiz olarak sunulur. Ancak, VHDL kullanımı ile ilgili bazı araçlar ve entegre geliştirme ortamları her ne kadar ücretli olabilse de firmalar nispeten küçük uygulamalar için ücretsiz araçlar sağlıyorlar. Projede Intel'in ücretsiz Quartus ve AMD'nin ücretsiz Vivado uygulamaları kullanılmıştır.

Şekil 2.2 Intel Quartus logo

2.3. FPGA

FPGA, elektronik devrelerin tasarımında kullanılan bir entegre devre tipidir. FPGA'lar, içerdikleri programlanabilir mantık blokları, bellek öğeleri ve bağlantı kaynakları sayesinde özelleştirilebilir donanım geliştirmek için kullanılır.

FPGA yapılandırması genellikle uygulamaya özel tümleşik devre için kullanılana benzer bir donanım tanımlama dili kullanılarak belirtilir. Bir FPGA'nın mantık blokları karmaşık kombinasyonel işlevleri yerine getirecek şekilde yapılandırılabilir veya AND ve XOR gibi basit mantık kapıları olarak hareket edebilir. Çoğu FPGA'da mantık blokları, basit flip-floplar veya bellek blokları içerir.

2.3.1. Kullanıldığı Yerler

Bir FPGA hesaplanabilen herhangi bir problemi çözmek için kullanılabilir. FPGA'lar, geniş bir uygulama yelpazesinde kullanılır; özellikle dijital sinyal işleme, görüntü işleme, ağ iletişimi ve gömülü sistemlerde yaygındır.

FPGA'ların kullanımındaki bir başka eğilim de, bir algoritmanın belirli bölümlerini hızlandırmak ve hesaplamanın bir kısmını FPGA ile genel bir işlemci arasında paylaşıldığı bir donanım hızlandırmadır.

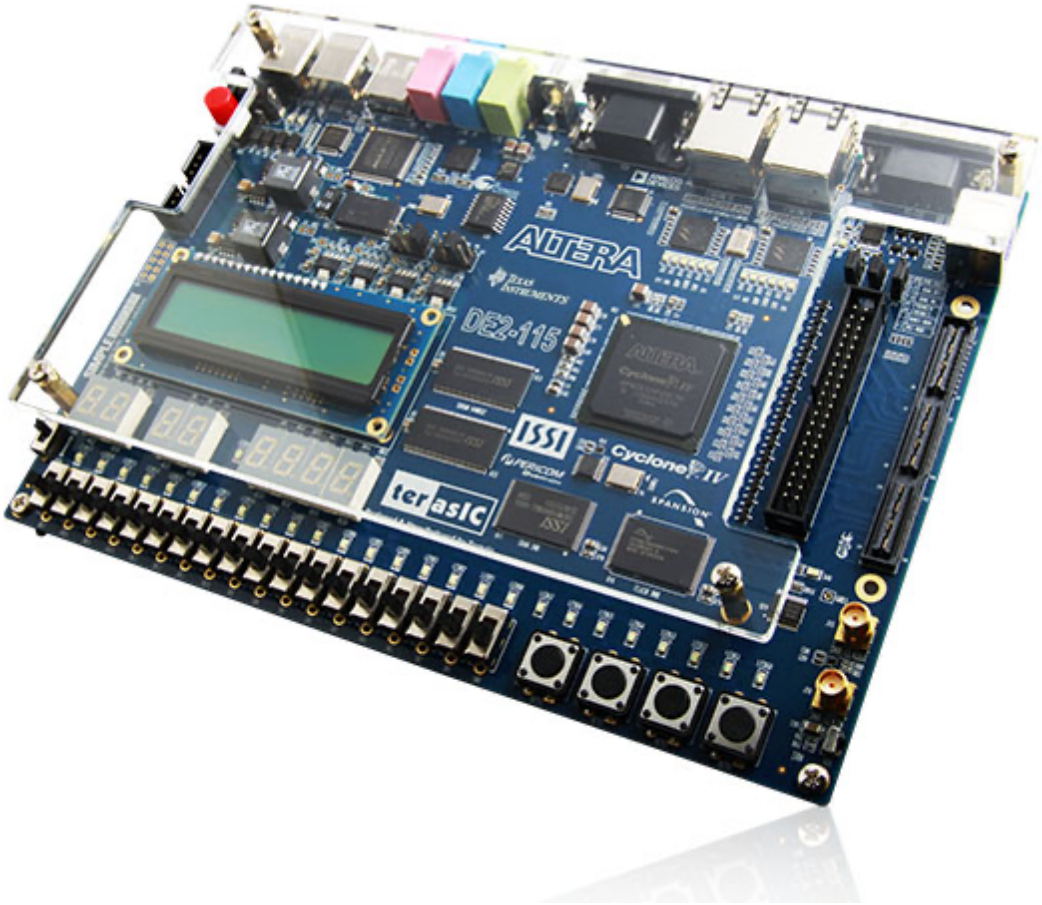
2.3.2. Neden FPGA

Tasarımcılar, FPGA'ları yüksek paralel işleme ve hızlı prototipleme yetenekleri için tercih eder. Bu cihazlar, geliştirme süreçlerini hızlandırırken aynı zamanda enerji verimliliği ve maliyet avantajları sunar.

FPGA'lar, endüstri standartlarına uygun olarak özelleştirilebilir donanım platformları sağlayarak, çeşitli projeler için esnek ve güçlü bir çözüm sunar. Bu nedenlerden dolayı da tasarlanan işlemcinin çalışabilmesi için mükemmel bir adaydır. Eğer bir donanımda sonradan bir hata farkedilirse ya da donanıma yeni bir işlev kazandırılmak isteniyorsa sadece FPGA içindeki kodu güncellemek yeterlidir.

2.3.3. Maliyet

FPGA'ların maliyeti, bir dizi faktöre bağlı olarak değişir. FPGA'lar genellikle model, kapasite, performans ve üreticiye göre fiyatlandırılır. Daha yüksek kapasiteli ve daha hızlı FPGA'lar genellikle daha pahalıdır. Biz ise bu projede bize okul tarafından ücretsiz olarak sağlanan $423 değerindeki Altera DE-2 115 kullanmaktayız.



Şekil 2.3 Örnek Altera DE2-115

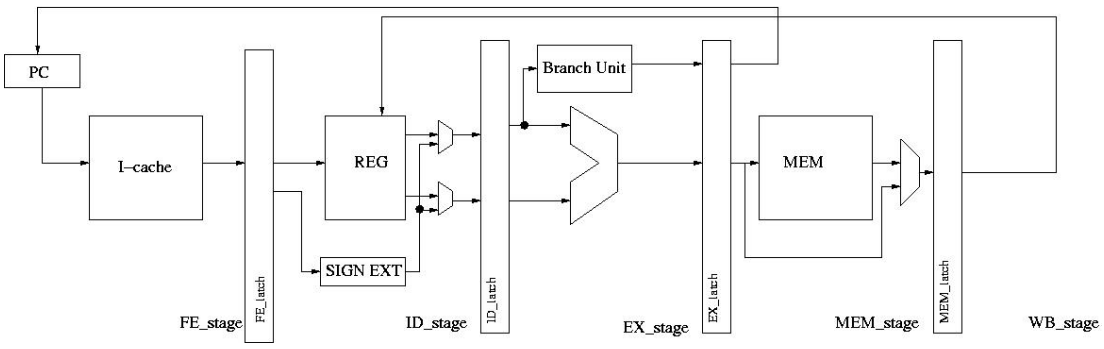
BÖLÜM 3. KULLANILAN YÖNTEMLER

Bu proje, boru hattı, dallanma öngörüsü ve veri yönlendirmesi gibi önemli birkaç yöntemi içeriyor. Boru hattı, işlemci performansını artırmak için görevleri parçalayan ve eş zamanlı olarak yürüten bir yöntemdir. Dallanma öngörüsü, işlemcideki dallanma komutlarının gelecekteki buyrukları tahmin etme yeteneğini ifade eder. Veri yönlendirmesi ise, işlemci içinde boru hattının neden olduğu veri bağımlılıklarını minimize ederek performansı artıran bir tekniktir.

Bu yöntemlerin kullanımı, işlemcinin işlemleri daha hızlı ve etkili bir şekilde gerçekleştirmesini sağlamak adına önemli bir rol oynar. Boru hattı, işlemci tasarımında paralel işleme yeteneklerini artırarak performans kazançları elde etmeyi hedefler. Dallanma öngörüsü, işlemcideki dallanma komutlarının doğru bir şekilde tahmin edilmesini ve bu sayede boşa geçirilen döngülerin azaltılmasını amaçlar.

Veri yönlendirmesi ise, işlemcideki veri bağımlılıklarını minimumda tutarak veri erişim sürelerini optimize eder. Bu yöntemlerin bir arada kullanılması, işlemcinin genel performansını artıracaktır.

3.1. Boru Hattı



Şekil 3.1 Örnek RISC Boru Hattı Yapısı

Boru hattı, bilgisayar sistemlerinde paralelizmi veya eşzamanlılığı içselleştirmenin bir formudur. Bu, bir hesaplama işleminin (örneğin, bir buyruğun) birkaç alt sürece (boru hattı segmentleri) bölünmesini ve bunların özel bağımsız üniteler tarafından

yürütülmesini ifade eder. Ardışık işlemler (buyruklar), endüstriyel bir montaj hattına benzer şekilde örtüşen bir modda gerçekleştirilebilir. Bu nedenle, çok genel bir şekilde ifade edilirse, boru hattı, tekrarlanan bir sıralı işlemi, her biri diğerleriyle eşzamanlı olarak çalışabilen özel bağımsız bir modülde etkili bir şekilde yürütme tekniği olarak tanımlanabilir [7]. Genellikle beş aşamalı (getir, çöz, yürüt, bellek ve geri yaz) bir boru hattı yapısı kullanılır.
İlk aşama, işlemci tarafından bellekten bir buyruğun alınmasını sağlar. Ardından, çözümleme aşaması, alınan buyruğun çözümlenmesi ve gerekli verilerin belirlenmesi işlevini yerine getirir.

Yürütme aşamasında, işlemcinin AMB veya özel işlev birimleri tarafından gerçekleştirilen hesaplamalar gerçekleşir. Yazma aşamasında, sonuçlar belleğe yazılır. Son olarak, geri alma aşamasında ise yazmaçlara veri yazılır. Boru hattı tasarımı, işlemci performansını artırarak aynı anda birden çok buyruğun yürütülmesine izin verir, ancak veri bağımlılıkları ve dallanma gibi karmaşıklıkları yönetmek için dikkatli bir tasarım gerektirir.

Bu şekilde, işlemciler daha hızlı ve verimli çalışabilir, ancak karmaşıklıkları yönetme ve hata durumlarını ele alma konularında dikkatli bir tasarıma ihtiyaç duyar.

3.1.1. Avantajları

- Paralel İşleme ve Hız Artışı:

Boru hattı, buyrukları paralel aşamalarda yürüterek toplam işlem hızını artırır. Bu, işlemci saat hızını optimize ederek daha fazla işlemin daha kısa sürede tamamlanmasına olanak tanır.

- Yeniden Kullanılabilirlik ve Modüler Tasarım:

Boru hattı mimarisi, aynı aşamaları birçok farklı buyrukta kullanabilir. Bu, tasarımın modüler ve yeniden kullanılabilir olmasını sağlar, böylece farklı işlemci modelleri için tasarım sürecini hızlandırabilir.

- Verimli Kaynak Kullanımı:

Boru hattı sayesinde, işlemci aynı anda birden çok buyruğu yürütebilir, bu da kaynakların daha verimli bir şekilde kullanılmasını sağlar.

3.1.2. Dezavantajları

- Veri Bağımlılıkları ve Bekleme Süreleri:

Boru hattında bir aşamada meydana gelen hata veya gecikme diğer aşamaları da etkileyebilir. Veri bağımlılıkları ve beklemeler nedeniyle boru hattının verimliliği düşebilir.

- Karmaşıklık ve Hata Ayıklama:

Boru hattı tasarımı, karmaşık denetim birimi ve aşama geçişleri içerir. Bu durum, tasarımın karmaşıklığını artırır ve hata ayıklama sürecini zorlaştırabilir.

- Fiziksel Gerçekleştirmenin Maliyeti:

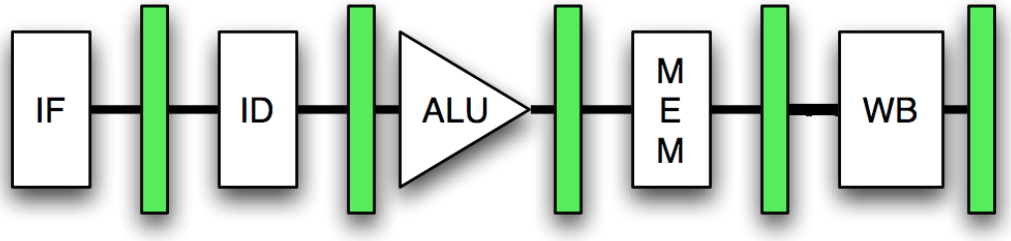
Boru hattı tasarımının fiziksel gerçekleştirmesi genellikle daha maliyetlidir. Bu durum, ek donanım ve üretim maliyetlerini artırabilir.

3.1.3. Özet

Boru hattı mimarisi, işlemcilerin günümüzdeki yüksek performans ve hızlı işleme beklentilerini karşılamak için yaygın olarak kullanılan etkili bir stratejidir. Ancak, avantajlarına rağmen veri bağımlılıkları, karmaşıklık ve maliyet gibi dezavantajları da göz önünde bulundurulmalıdır.

Yine de, doğru bir şekilde uygulandığında, boru hattı, bilgisayar sistemlerinde paralel işleme yetenekleri ve modüler tasarım avantajları ile geniş bir kullanım alanına sahiptir.

3.1.4. Kullanım Şekli



Şekil 3.2. Boru Hattı Şeması [8]

Tasarlanan işlemcinin içinde birçok farklı birim bulunmaktadır. Bunlar 5 aşamalı bir boru hattına uygun olacak şekilde yerleştirilmiştir. Aşamalar sırasıyla şu şekildedir: Getir, Çöz, Yürüt, Bellek ve Geri Yaz

3.1.4.1. Getir

İşlemci bu aşamada program sayacı aracılığıyla buyruk belleğinden uygun buyruğu alır. Uygun buyruğu aldıktan sonra dallanma öngörücüsüne buyruğun bazı bitlerini aktarar buyruk hakkında ona bilgi aktarır. Eğer buyruk bir dallanma, atlama veya bellek okuma buyruğu değilse sadece program sayacının değerini 4 arttırır.

Eğer buyruk dallanma buyruğu ise içindeki algoritmaya göre bir tahmin yapar ve ona dallanır. Eğer bu tahmin hatalı ise dallanmadan sonraki boru hattına sokulan buyrukları geçersiz yapan bir sinyal üretir. Ondan sonra yanlışını düzeltip diğer tarafa dallanır. Başarısız dallanma, işlemcinin 3 tane gereksiz buyruğu içine almasına neden olur. Yani 3 saat darbesi süresi vakit çöp olmuş olur.

2 farklı atlama buyruğu bulunmaktadır. Eğer JAL buyruğu gelirse içindeki anlık değeri direkt olarak program sayacına aktarır. JALR buyruğu gelirse atlanılacak adersin hesaplanması gerekecektir. Bu hesaplama işleminden dolayı 2 saat darbesi süresince yeni buyruk işlemciye alınamayacaktır.

Boru hattında şöyle bir problem bulunmaktadır: İki adet buyruğumuz var diyelim. Birincisi bellek okuma buyruğu olsun. Bellekten okuduğu veriyi x10 yazmacına

yazsın. İkinci buyruksa birinci buyruktan hemen sonra gelen ve x10 yazmacını okumak isteyen bir buyruk olsun. Birinci buyruk 3.aşamadayken ikinci buyruk 2. aşamada olacaktır. Bu durumdayken ikinci buyruğun x10 yazmacının yeni değeri okuması imkansızdır. Çünkü birinci buyruk henüz 4.aşamada bulunan bellekten okuma işlemini gerçekleştirememiştir. Bundan dolayı veri yönlendirmesi de beyhudedir.

Bu problemi çözmek için işlemci bir bellek okuma buyruğundan sonra gelen yeni buyruğun ne olduğunu inceler. Eğer birisinin yazdığı yazmacı diğeri okuyorsa ikinci buyruğun ikinci aşamaya geçmesi engellenir. Böylece birinci buyruk ikinci buyruğun bir adım önüne geçirilmiş olur. İkinci buyruk 2.aşamadayken birinci buyruk 4.aşamada olacaktır. İkisi bu aşamalardayken veri yönlendirmesi yapılarak bu problem çözülmüş olur.

3.1.4.2. Çöz

Bu aşamada buyruğun ilgili bitleri alınarak buyruğun hem bu aşamada hem de ilerleyen aşamalarda ihtiyacı olduğu denetim sinyalleri üretilir. Ayrıca bu aşamadayken ihtiyacı olan buyruklar için yazmaç okuma ve anlık genişletme işlemleri gerçekleştirilir.

Gereken durumlarda, boru hattının üçüncü, dördüncü ve beşinci aşamalarının uygun olanından buraya veri yönlendirmesi yapılır. Hatta bu işlemi yapabilmek için özel bir birim oluşturulmuştur.

3.1.4.3. Yürüt

Veriler üzerinde asıl işlemlerin yapıldığı aşamadır. Aritmetik ve mantıksal işlemler yapan tüm buyrukların istedikleri hesaplamalar buradaki AMB üzerinde yapılır. Özellikle yeni eklenen kriptografi buyrukları için burası kritik bir noktadır. Herhangi bir buyruk AMB'yi kullanmadan önce denetim sinyalleri aracılığıyla işlenecek olan veriler seçilir sonrasıda yine denetim sinyalleri aracılığı ile uygun işlem seçilir.

Ayrıca AMB ile bellek okuma ve yazma buyrukları için adres hesabı da yapılmaktadır. Keza dallanma buyrukları için gereken karşılaştırma işlemleri de burada yapılmaktadır. Çıkan sonuçlar dallanma öngörücüsüne ayrıyeten iletilir.

Bilindiği üzeri RISC-V'de atlama buyrukları kendinden sonraki buyruğun adresini bir yazmaçta saklarlar. Bu buyrukların ihtiyaç duydukları adres hesaplaması (PC + 4) da AMB üzerinden yapılır. JALR buyruğu için gereken adres hesaplaması ilave bir toplayıcı üzerinden yapılmaktadır. İlaveten söylemek gerekirse AUIPC buyruğu burada bir ADDI vs. gibi buyruk gibi kabul edilip PC + Imm işlemi de AMB'de yapılır.

3.1.4.4. Bellek

Bu aşama sadece bellek okuma yazma işlemi yapan buyrukları için anlamlıdır. Burada bulunan belleğin belirtilen adresinden denetim sinyali aracılığıyla 8, 16 veya 32 bitlik veriler okunabilmektedir. Ayrıca yine aynı boyutlarda belirtilen adrese yazma işlemleri de gerçekleştirilmektedir. Keza okunan veya yazılan veriler 8 veya 16 bitse işaretli ya da işaretsiz olarak okunup yazılabilmektedir.

3.1.4.5. Geri Yaz

Bu aşamanın amacı önceki aşamalarda elde edilen verinin doğru yazmaca kaydedilmesidir. Denetim sinyalleri aracılığıyla uygun değer alınır. Sonrasında bu değer belirtilen yazmaca aktarılır.

3.2. Dallanma Öngörüsü

İşlemcilerdeki dallanma öngörüsü, bilgisayar mimarisi tasarımında kilit bir optimizasyon stratejisidir. Eğer geliştirilen işlemcide bir boru hattı sistemi konmuşsa performansı arttırmak için bu yöntemi kullanmak çok elzemdir.

Yüksek performanslı bilgisayar sistemlerinde, koşullu dallanma talimatlarından kaynaklanan performans kayıpları, bir dallanma sonucunu öngörerek ve gerçek

sonuç bilinmeden önce sonraki talimatları getirerek, çözümleyerek ve/veya işleyerek en aza indirilebilir. [7]

İşlemcilerin nereye dallanacğı bazen hemen hesaplanamayabilir. Hesaplanma sürecinde işlemcinin normalde beklemesi gerekir. Hesaplama bittikten sonra işlemci içine yeni buyruklar almaya devam eder. Dallanma öngörüsü ile bu beklemenin önünce geçmeyi hedefliyoruz. Tipik bir işlemci, bir komutu işlerken bir sonraki komutu tahmin eder ve bu öngörü sonucunda hesaplanan yola devam eder.

Eğer öngörü doğruysa, işlemci normal bir şekilde çalışır ve işlemcinin performansı artar. Ancak, yanlış bir öngörü durumunda, işlemci dallanma tahmin hatalarını yönetmek için özel tasarlanmış teknikler kullanır.

3.2.1. Avantajlar

- Performans Artışı:

Dallanma öngörüsü, işlemci performansını artırarak programların daha hızlı bir şekilde yürütülmesine olanak tanır. Özellikle döngüler, koşullu ifadeler ve diğer dallanma komutları gibi yapılar, öngörü algoritmaları sayesinde daha etkili bir şekilde işlenebilir.

- Enerji Verimliliği:

Doğru bir şekilde uygulandığında, dallanma öngörüsü enerji verimliliğini artırabilir. Çünkü öngörü algoritmaları, gereksiz işlemleri önleyerek enerji tüketimini optimize edebilir.

3.2.2. Dezavantajlar

- Yanlış Tahminler:

Dallanma öngörüsü, her zaman doğru tahmin yapamaz. Özellikle karmaşık ve değişken yapıya sahip programlarda yanlış tahminlerin sıkça olması, performans avantajını azaltabilir.

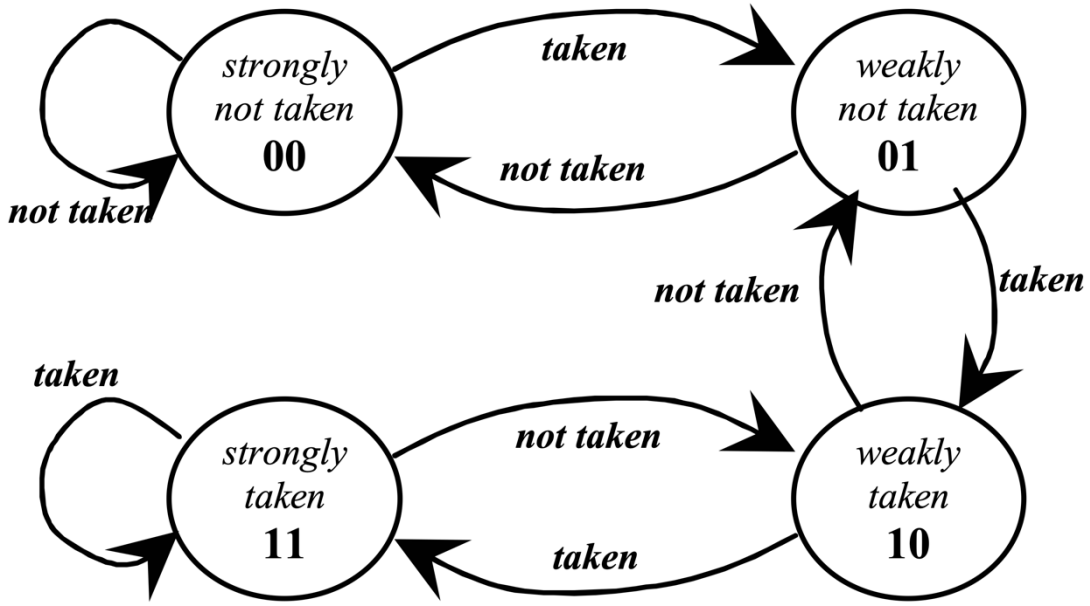
- Donanım Karmaşıklığı:

Gelişmiş dallanma öngörü algoritmaları, işlemci içinde daha karmaşık donanım birimlerini gerektirebilir. Bu durum, işlemcinin tasarımını karmaşıklaştırabilir ve üretim maliyetini artırabilir.

3.2.3. Özet

Dallanma öngörüsü, işlemcilerin karmaşık program yapılarıyla daha etkili bir şekilde başa çıkmasına olanak tanıyan kritik bir optimizasyon stratejisidir. Ancak, avantajlarına rağmen yanlış tahminler ve donanım karmaşıklığı gibi dezavantajlarla da karşılaşabilir. Doğru bir şekilde uygulandığında, performans artışı ve enerji verimliliği gibi avantajlar, bu teknikten elde edilen faydalar arasında yer alır.

3.2.4. Kullanım Şekli



Şekil 3.3 Çift Doruklu Dallanma Öngörüsü [9]

Yazılan programlar genelde bir çok döngü içermektedir. Sistem eğer bir döngünün içeresine girdi ise sürekli olarak dallanması gerekir. Başka bir örnek olarak “if” yapılarından verilebilir. Yazılımcılar sıklıkla “if” yapılarını istisna durumlar için kullanırlar. Bundan dolayı da işlemci, “if” bloğunun içerisine girmeden atlaması gerekir.

Dallanma tahmini, her dallanma için sabit bir sonucu tahmin eden statik teknikler ve kontrol akışının geçmişini analiz ederek gelecekteki davranışını öngören dinamik teknikleri içerir [9]. Bu tür dallanmaları daha rahat anlayabilmek amacıyla işlemcide bir dinamik dallanma öngörüsü sistemi olan çift doruklu dallanma öngörüsü kullanılmaktadır. Bu mekanizmada sistemin önceki 2 dallanma sonucu saklanır ve bunun üzerinden öngörülerde bulunulur. Bu sistemi 4 durumlu bir durum makinesi olarak düşünülebilir. Durumları şu şekilde adlandırabiliriz: “Emin – Dallan”, “Şüpheli – Dallan”, “Şüpheli – Dallanma” ve “Emin – Dallanma”

- Emin – Dallan (11): Daha önceki 2 dallanma sonucu da “dallan” olarak çıktığı durumdur. Bu durumdayken “dallan” şeklinde tahmin yürütür. Eğer tahmin doğru çıkarsa bulunduğu durum değişmez ama yanlış çıkarsa “Belirsiz – Dallan” durumuna geçer.
- Belirsiz – Dallan (10): İki önceki dallanma sonucunun “dallan”, bir önceki dallanma sonucunun ise “dallanma” olarak çıktığı durumdur. Bu durumdayken “dallan” şeklinde tahmin yürütür. Eğer tahmin doğru çıkarsa bulunduğu durum “Emin Dallan” durumuna geçer ama yanlış çıkarsa “Belirsiz – Dallanma” durumuna geçer.
- Belirsiz – Dallanma (01): İki önceki dallanma sonucunun “dallanma”, bir önceki dallanma sonucunun ise “dallan” olarak çıktığı durumdur. Bu durumdayken “dallanma” şeklinde tahmin yürütür. Eğer tahmin doğru çıkarsa bulunduğu durum “Belirsiz – Dallan” durumuna geçer ama yanlış çıkarsa “Emin – Dallanma” durumuna geçer.
- Emin – Dallanma (00): Daha önceki 2 dallanma sonucu da “dallanma” olarak çıktığı durumdur. Bu durumdayken “dallanma” şeklinde tahmin yürütür. Eğer tahmin doğru çıkarsa bulunduğu durum değişmez ama yanlış çıkarsa “Belirsiz – Dallanma” durumuna geçer.

Bu mekanizma ayrıca iki bitlik bir sayaç olarak da düşünülebilir. Dallanma sonucu her “dallan” olarak çıktığında 1 arttırıp, her “dallanma” olarak çıktığında ise 1 azaltarak da bu sistem oluşturulabilir. Zaten projede de bu mantık kullanılacaktır.

3.3. Veri Yönlendirmesi

Boru hattına sahip olan işlemcilerde veri sorunlarını etkin bir şekilde çözmek için kullanılan bir yöntemdir. Veri yönlendirmesi, işlemcinin veri yolu üzerindeki veri akışını optimize ederek performans artışı sağlar.

Tipik bir boru hattına sahip işlemcide, bir buyruk kendinden önce gelen buyruğun oluşturacağı veriyi kullanmak istediğinde onu beklemeye mecbur kalır. Veri yönlendirmesi ile bu tür gecikmeleri azaltmak veya tamamen kaldırmak elimizdedir.

3.3.1. Avantajlar

- Veri Bağımlılıklarını Azaltma:

Veri yönlendirmesi, işlemcideki veri bağımlılıklarını minimize ederek işlemlerin daha hızlı bir şekilde tamamlanmasını sağlar. Bu, işlemcinin veriye daha hızlı erişmesini ve kullanmasını mümkün kılar.

- Performans Artışı:

Veri yönlendirmesi, işlemcilerin belirli bir aşamada kullanılacak verileri önceden belirlenmiş yollarla taşımasını sağlar. Bu sayede işlemci, verilere erişimde daha hızlı olabilir ve toplam performans artışı sağlayabilir.

3.3.2. Dezavantajlar

- Karmaşıklık ve Hata Ayıklama:

Veri yönlendirmesinde farklı birimler arası bağlantılar çekmek zorunda kalınır. Böylece işlemcinin tasarımını karmaşıklaştırır. Ayrıca karmaşıklaşan tasarım nedeniyle sistemdeki hataların ayıklanaması zorlaşır.

3.3.3. Özet

Veri yönlendirmesi, işlemcilerin veri bağımlılıklarını azaltarak performans artışı ve enerji verimliliği sağlayan önemli bir tasarım stratejisidir. Ancak, karmaşıklık göz önünde bulundurulmalıdır. Doğru bir şekilde uygulandığında, veri yönlendirmesi, işlemcilerin veri erişim işlemlerini optimize ederek genel sistem performansını artırabilir

3.3.4. Kullınım Şekli

Boru hattındaki veri sorununu çözmek için işlemciye bir "Veri Yönlendirmesi" mekanizması inşa edildi. 2'nci aşamadaki buyruğun kaynak yazmacı, 3'ncü, 4'üncü veya 5'inci aşamadaki buyruklar için hedef yazmacı olabilir. Bu durum buyruğun yazmacın eski değerini okumasına neden olur. Bundan dolayı bu mekanizma 2'nci aşamadaki buyruğun kullandığı kaynak yazmacını hedef yazmacı olarak kullanan 3'ncü, 4'üncü veya 5'inci aşamada bulunan buyrukların ürettiği veriyi, 2'nci aşamada bulunan buyruğa aktarılmasını sağlar.

Fakat bu verilerden sadece bir tanesinin aktarılması gerekir. 2'nci aşamadaki buyruk için yazmacın güncel değeri olması gerekir. Yazmacı en son değiştiren buyruk boru hattının sol tarafına yakın olacağı için veri yönlendirmesi yapıldığında sol tarafa yakın olan buyruktan gelen veriler seçilir.

Örneğin 2'inci aşamadaki buyruğun kaynak yazmacı 3'ncü ve 5'inci aşamadaki buyruklar tarafından hedef yazmacı olarak kullanılsın. Böyle bir durumda sistem 3'üncü aşamadaki buyruktan gelen verileri kullanacaktır.

Hedef yazmacına boru hattındaki 3 farklı birimden veri gelir. Bunlar şu birimlerdir:

- Anlık Üreteci: “LUI” buyruğunda genişletilen anlık, direkt olarak hedef yazmacına yazılır. Diğer hiçbir buyruk için hedef yazmacına bu birimden veri gelmez.
- Veri Belleği: Bellek okuma buyruklarının hepsi hedef yazmacını buradan gelen veri ile doldurur.
- Aritmetik Mantık Birimi: “LUI” ve bellek okuma buyrukları dışındaki tüm hedef yazmacı kullanan buyruklar burada üretilen verileri hedef yazmacına yazar.

Boru hattının farklı aşamalarında, yönlendirilen verilerin kaynağı değişiklik göstermektedir. Bunlar şöyledir:

- 3’üncü Aşamadan Yönlendirilebilen Verilerin Kaynakları:
  - AMB
  - Anlık Genişletici
- 4’üncü Aşamadan Yönlendirilebilen Verilerin Kaynakları:
  - AMB
  - Anlık Genişletici
  - Veri Belleği
- 5’inci Aşamadan Yönlendirilebilen Verilerin Kaynakları:
  - AMB
  - Anlık Genişletici
  - Veri Belleği

Daha önce 3’üncü aşamadaki bellek buyruklarından veri yönlendirmesi yapılamayacağını bundan dolayı da bazı durumlarda yeni buyruk alımı duraklatıldığı söylenmişti. Bunun nedenini yukarıdaki veri kaynakları listesi bir kez daha göstermiş oldu.

Veri yönlendirmesi 2’nci aşamada bulunan veri yönlendiricisi aracılığıyla yapılır. Veri yönlendiricisi ileriki aşamaların hepsin hedef yazmacına yazılacak verinin dışında 2 farklı bilgi daha alır:

- Hedef Yazmacı Adresi

- Veri Yönlendirme Onayı

Hedef yazmacı adresi, 2'nci aşamadaki buyruğun kaynak yazmacı ile yönlendirilecek verinin yazılacağı yazmacın aynı olup olmadığının tespiti için kullanılır.

Veri yönlendirme onayı, o aşamadaki buyruğun veri yönlendirmesine uygun olup olmadığını belirtir. Bu onay sinyali aşağıdaki 3 sinyalin "VE" işlemine tâbi tutulmasıyla oluşur:

- Buyruk Aktifliği: Bu sinyal boru hattındaki bütün aşamalar boyunca taşınır. Eğer dallanma öngörüsü yanlış olursa uygun olan buyruklar için bu sinyal terslenir. Böylece bu buyrukların yazma yetkileri ve veri yönlendirebilme yetenekleri ellerinden alınır.
- Hedef Yazmacına Yaz: Bu sinyal denetim birimi tarafından oluşturulur. Eğer bu sinyal "1" ise buyruğun hedef yazmacına bir şey yazmayı amaçladığı anlaşılır.
- Hedef Yazmacına Yazılacak Veri Seçimi: Bu sinyal de denetim birimi tarafından oluşturulur ve asıl görevi 5'inci aşamada hedef yazmacına doğru verinin yazılmasını sağlamaktır. Bunun yanında son 3 aşamadaki veri yönlendirmesi için kullanılacak veriyi seçmeye de yarar. AMB ve anlık genişletici için sırasıyla "01" ve "11" değerlerini alır. Bellek okuma buyrukları için ise "00" değerlerini alır. Eğer bu aşamada bir bellek okuma buyruğu varsa henüz yönlendirebilecek bir verisi olmadığı için hatalı bir yönlendirmeye sebep olacaktır. Dikkat edilirse sağ biti "0" olan bir tek bellek okuma buyrukları bulunmaktadır. Bu sinyalin sağ biti kullanılarak buyruğun bellek okuma buyruğu olup olmadığı anlaşılır. Bundan dolayı sadece 3'ncü aşamada veri yönlendirme onayı için sağ biti kullanılır.

Not: Daha öncesinde bellek okuma buyruğunun hedef yazmacını kaynak yazmacı olarak kullanan bir buyruk bellek okuma buyruğundan hemen sonra gelirse diğer buyruğun geciktirileceğinden bahsedilmişti. Bundan dolayı 3'ncü aşamada bellek okuma buyruğunun algılanıp veri yönlendirmesinin engellenmesi anlamsız gözükebilir. Bellek okuma buyruğundan sonraki buyruk geciktirildiğinde boru hattının 3'ncü aşamasına geçen bellek okuma buyruğu 2'nci aşamada hala kalır. Bir nevi bellek okuma buyruğu kopyalanması gibi bir yan etki oluşur. Her iki buyruk da

aynı bellek adresini okuyup aynı yazmaca yazacağı için sistemde bir tutarsızlığa neden olmayacaktır. Fakat geciktirilen buyruğun önünde kopya bir bellek buyruğu olduğu için veri yönlendirilmesinin engellenmesi zorunludur.

BÖLÜM 4. İŞLEMCİDEKİ BUYRUKLAR

RISC-V'de buyrukların kullandığı yazmaçların ve anlıkların amacını göstermek için "rs1", "rs2", "rd" ve "imm" tabirleri kullanılır. Bunlar şu anlamlara gelmektedirler:

rs1: register source 1	→	1. Kaynak Yazmacı
rs2: register source 2	→	2. Kaynak Yazmacı
rd: register destination	→	Hedef Yazmacı
imm: immediate	→	Anlık

Özellikle "imm", köşeli parantez ile kullanılır. Köşeli parantezin içi anlığın bitlerini ifade etmektedir. Örneğin, imm[11:0] anlığın 0'ıncı bitinden 11'inci bitine, 11'de dahil, kadar ki tüm bitleri ifade eder. Bunun yanında anlığın işaretli bir sayı olduğu unutulmamalıdır.

RISC-V'in çevirici dilinde yazmaçları ifade etmek için x3, x6, x14 vs. yazılır. Örneğin, x3 ile 3. yazmaç ifade edilir. RISC-V'de 32 tane 32 bitlik yazmaç olduğundan yazmaçlar x0'dan x31'e kadar ifade edilebilir. Bunlarda x0 yani sıfırıncı yazmacın değeri her zaman sıfırdır. Asla değiştir

Ayrıca 3 bitlik "funct3" ve 7 bitlik "funct7" ile opcode dışındaki buyrukların işlev bitleri kastedilmektedir.

RISC-V'deki bütün buyrukların uzunlukları sabit ve 32 bittir. İstenirse ek buyruk komutları ile 16 bitlik buyruklar da desteklenebilir.

RV32-I’da 6 tip buyruk bulunmaktadır. Bunlar şunlardır:

R-Tipi (Register)

I-Tipi (Immediate)

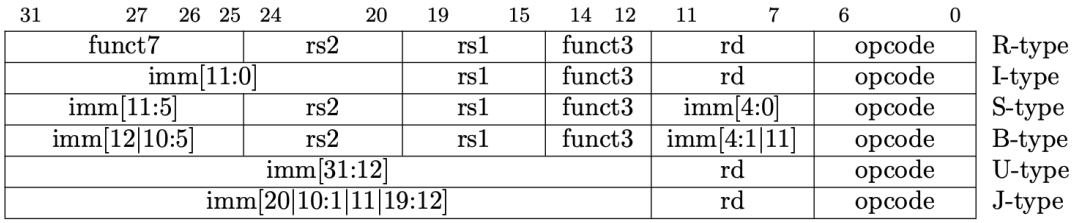
S-Tipi (Store)

B-Tipi (Branch)

U-Tipi (Upper)

J-Tipi (Jump)

Bunların bit dizilimleri ise şu şekildedir:



Şekil 4.1 Buyruk Tipleri

RISC-V’de buyruk adreslerinin hizalanmış olması istenmektedir. Yani adresin 4’e bölünebilir olması gerekir. Binaenaleyh dallanma/atlama ile alakalı olan buyruklarda adrese eklenecek olan anlık 1 kere sola kaydırılmış olarak kabul edilir. Bundandır ki yukarıdaki görseldeki buyrukların bazılarında imm[0] bulunmaz. Çünkü anlık sola kaydırılmış kabul edildiği için imm[0] = 0 olmak zorundadır.

Buyrukların 4’e bölünebilmesi gerektiğinde dolayı aslında anlığında 4’e bölünebilir olması gerekir. Fakat burada sadece 2’ye bölünebilir alır. Çünkü aynı zamanda ek buyruk kümeleri ile de sıkıştırılmış buyruklar, 16 bitlik buyruklar, desteklenir. Burada sıkıntı çıkmaması için 2 kere değil 1 kere sola kaydırılmış anlık alınır.

4.1. R-Tipi Buyruklar

Bu buyruklar kaynak yazmaçlardaki veriyi alıp aritmetik veya mantıksal işlemlerden birini yapar. Sonucu da hedef yazmaca yazarlar. Genel biçimleri şöyledir:

→ (Buyruk Adı) rd, rs1, rs2

Bu tipte toplam 10 adet buyruk vardır:

ADD: Addition
→ rs1 ile rs2'yi toplar sonucu rd'ye yazar.
→ ADD x6, x25, x30

SUB: Subtract
→ rs1 ile rs2'yi birbirinden çıkarır sonucu rd'ye yazar.
→ SUB x1, x3, x6

SLL: Shift Left Logical
→ rs1'i rs2'deki değer kadar sıfırlar ile sola kaydırır sonucu rd'ye yazar.
→ SLL x4, x22, x1

SRL: Shift Right Logical
→ rs1'i rs2'deki değer kadar sıfırlar ile sağa kaydırır sonucu rd'ye yazar.
→ SRL x20, x3, x11

SRA: Shift Right Arithmetic
→ rs1'i rs2'deki değer kadar işaret biti ile sağa kaydırır sonucu rd'ye yazar.
→ SRA x14, x13, x13

SLT: Set Less Than
→ rs1 ile rs2 işaretlerine bakılacak şekilde karşılaştırılır. Eğer rs1, rs2'den küçükse rd'ye 1, değilse 0 yazar.
→ SLT x11, x22, x5

SLTU: Set Less Than Unsigned

→ rs1 ile rs2 işaretlerine bakılmadan karşılaştırılır. Eğer rs1, rs2'den küçükse rd'ye 1, değilse 0 yazar.

→ SLTU x19, x3, x5

XOR

→ rs1 ile rs2'yi ÖZEL VEYA işlemine tâbi tutar ve sonucu rd'ye yazar.

→ XOR x25, x12, x31

OR

→ rs1 ile rs2'yi VEYA işlemine tâbi tutar ve sonucu rd'ye yazar.

→ OR x6, x5, x0

AND

→ rs1 ile rs2'yi VE işlemine tâbi tutar ve sonucu rd'ye yazar.

→ AND x3, x5, x6

4.2. I-Tipi Buyruklar

Bu buyrukları yaptıkları işe göre 3 kısma ayırabiliriz:

4.2.1. İşlem Buyrukları

İşlem buyrukları rs1'deki veriyi ve anlığı alıp aritmetik veya mantıksal işlemlerden birini yapar. Sonra sonucu hedef yazmaca yazar. Genel biçimleri şöyledir:

→ (Buyruk Adı) rd, rs1, imm

Bu tipte toplam 9 adet buyruk vardır:

ADDI: Addition Immediate

→ rs1 ile anlığı toplar sonucu rd'ye yazar.

→ ADDI x4, x6, A1D

SLLI: Shift Left Logical Immediate

→ rs1'i anlıkdaki değer kadar sıfırlar ile sola kaydırır sonucu rd'ye yazar.

→ SLLI x22, x28, 001

SRLI: Shift Right Logical Immediate

→ rs1'i anlıkdaki değer kadar sıfırlar ile sağa kaydırır sonucu rd'ye yazar.

→ SRLI x10, x10, 003

SRAI: Shift Right Arithmetic Immediate

→ rs1'i anlıkdaki değer kadar işaret biti ile sağa kaydırır sonucu rd'ye yazar.

→ SRAI x10, x13, 00A

SLTI: Set Less Than Immediate

→ rs1 ile anlığın işaretlerine bakılacak şekilde karşılaştırılır. Eğer rs1, anlıkdan küçükse rd'ye 1, değilse 0 yazar.

→ SLTI x23, x21, 365

SLTIU: Set Less Than Immediate Unsigned

→ rs1 ile anlığın işaretlerine bakılmadan karşılaştırılır. Eğer rs1, rs2'den küçükse rd'ye 1, değilse 0 yazar.

→ SLTIU x1, x21, 111

XORI: XOR Immediate

→ rs1 ile anlığı ÖZEL VEYA işlemine tâbi tutar ve sonucu rd'ye yazar.

→ XORI x26, x25, A45

ORI: OR Immediate

→ rs1 ile anlığı VEYA işlemine tâbi tutar ve sonucu rd'ye yazar.

→ XORI x14, x17, 236

ANDI: AND Immediate

→ rs1 ile anlığı VE işlemine tâbi tutar ve sonucu rd'ye yazar.

→ ANDI x4, x19, 9C5

4.2.2. Bellek Buyrukları

Bellek buyrukları ise rs1 ile anlığı toplayıp bu toplamın işaret ettiği adresteki veriyi bellekten alır. Sonra bu değeri rd'ye yazar. Genel biçimleri şöyledir:

→ (Buyruk Adı) rs2, offset(rs1) // offset = imm

Bu tipte toplam 5 adet buyruk vardır:

LB: Load Bayt
→ rs1 ile anlığı toplayıp bu toplamın işaret ettiği adresteki verinin 1 baytını bellekten alır. Sonra işaret biti ile genişletip rd'ye yazar.
→ LB x14, 004(x13)

LBU: Load Bayt Unsigned
→ rs1 ile anlığı toplayıp bu toplamın işaret ettiği adresteki verinin 1 baytını bellekten alır. Sonra sıfır ile genişletip rd'ye yazar.
→ LBU x31, AAA(x1)

LH: Load Halfword
→ rs1 ile anlığı toplayıp bu toplamın işaret ettiği adresteki verinin 2 baytını bellekten alır. Sonra işaret biti ile genişletip rd'ye yazar.
→ LH x4, A3D(x14)

LHU: Load Halfword Unsigned
→ rs1 ile anlığı toplayıp bu toplamın işaret ettiği adresteki verinin 2 baytını bellekten alır. Sonra sıfır ile genişletip rd'ye yazar.
→ LHU x28, 02A(x6)

LW: Load Word
→ rs1 ile anlığı toplayıp bu toplamın işaret ettiği adresteki verinin 4 baytını bellekten alır. Sonra rd'ye yazar.
→ LW x30, F12(x3)

4.2.3. Koşulsuz Dallanma Buyrukları

Bu buyruklar ile koşulsuz atlama işlemlerini gerçekleştirebilirler. Genel biçimleri şöyledir:

→ (Buyruk Adı) rd, rs1, imm

Bu tipte sadece 1 adet buyruk vardır:

JALR: Jump and Link Register
→ Program sayacının bir sonraki değeri eski konuma geri dönme imkanı için hedef yazmacına yazılır. Devamında kaynak yazmacındaki değer ile verilen anlık değer toplanır. Sonrasında çıkan sonucun belirttiği adrese dallanılır.

→ JALR x7, x25, AD2

4.3. S-Tipi Buyruklar

Bu buyrukların tek amacı yazmaçlardaki verilerin belleğe depolanmasını sağlamaktır. rs1'deki değer ile anlık toplanarak bir bellek adresi elde edilir. Adresin işaret ettiği yere rs2 yazmacındaki veri yazılır. Bu buyrukların genel biçimleri şöyledir:

→ (Buyruk Adı) rs2, offset(rs1) // offset = imm

Bu tipte toplam 3 adet buyruk vardır:

SB: Store Bayt
→ rs1 ile anlığı toplayıp bu toplamın işaret ettiği bellekteki adrese rs2'deki verinin 1 baytı yazılır.
→ SB x5, 0AC(x17)

SH: Store Halfword
→ rs1 ile anlığı toplayıp bu toplamın işaret ettiği bellekteki adrese rs2'deki verinin 2 baytı yazılır.
→ SH x17, FE7(x27)

SW: Store Word
→ rs1 ile anlığı toplayıp bu toplamın işaret ettiği bellekteki adrese rs2'deki verinin 4 baytı yazılır.
→ SW x11, A1E(x22)

Not: RISC-V özel olarak bir Endian tanımlamaz. Bu tamamen tasarlayan kişiye bırakılmıştır.

4.4. B-Tipi Buyruklar

Bu buyrukların tek amacı koşullu dallanmadır. 2 kaynak yazmaçtaki veriyi kıyaslarlar. Sonra kıyaslama doğru ise anlık, program sayacı ile toplanarak hedeflenen yere dallanılır. Genel biçimleri şöyledir:

→ (Buyruk Adı) rs1, rs2, offset // offset = imm

Bu tipte toplam 6 adet buyruk vardır:

BEQ: Branch (if) Equal
→ rs1'deki değer ile rs2'deki değer aynı ise dallanır değilse bir şey yapmaz.
→ BEQ x12, x13, 134

BNE: Branch (if) Not Equal
→ rs1'deki değer ile rs2'deki değer aynı ise bir şey yapmaz değilse dallanır.
→ BNE x13, 24, F43

BLT: Branch (if) Less Than
→ rs1'deki değer, rs2'deki değerden, işarete bakarak, küçük ise dallanır değilse bir şey yapmaz.
→ BLT x26, x15, EE3

BLTU: Branch (if) Less Than Unsigned
→ rs1'deki değer, rs2'deki değerden, işarete bakmadan, küçük ise dallanır değilse bir şey yapmaz.
→ BLTU x29, 11, ED1

BGE: Branch (if) Greater Equal (Than)
→ rs1'deki değer, rs2'deki değerden, işarete bakarak, büyük veya eşit ise dallanır değilse bir şey yapmaz.
→ BGE x20, x10, F1A

BGEU: Branch (if) Greater Equal (Than) Unsigned
→ rs1'deki değer, rs2'deki değerden, işarete bakmadan, büyük veya eşit ise dallanır değilse bir şey yapmaz.
→ BGEU x30, x31, ABB

4.5. U-Tipi Buyruklar

Dikkat edilirse diğer buyruklar sadece 12 bitlik bir anlık değer alıyorlar. Fakat yazmaçlarımızın 32 bitlik bir genişliği vardır. Bu aradaki 20 biti doldurmak için bu buyruklar kullanılır. Genel biçimleri şöyledir:

→ (Buyruk Adı) rd, imm

Bu tipte toplam 2 adet buyruk vardır:

LUI: Load Upper Immediate
→ Verilen 20 bitlik anlık hedef yazmacının üst 20 bitine yazılır.
→ X13, A2B3C

AUIPC: Add Upper Immediate (to) Program Counter
→ İlk önce verilen 20 bitlik anlığın düşük 12 bitine sıfır koyularak genişletilir. Sonra program sayacı ile toplanır. Sonuç hedef yazmacına yazılır. Bu komuttan sonra JALR buyruğu ile de 32 bitlik bir adrese atlanabilir.
→ AUIPC x22, BC2A1

4.6. J-Tipi Buyruklar

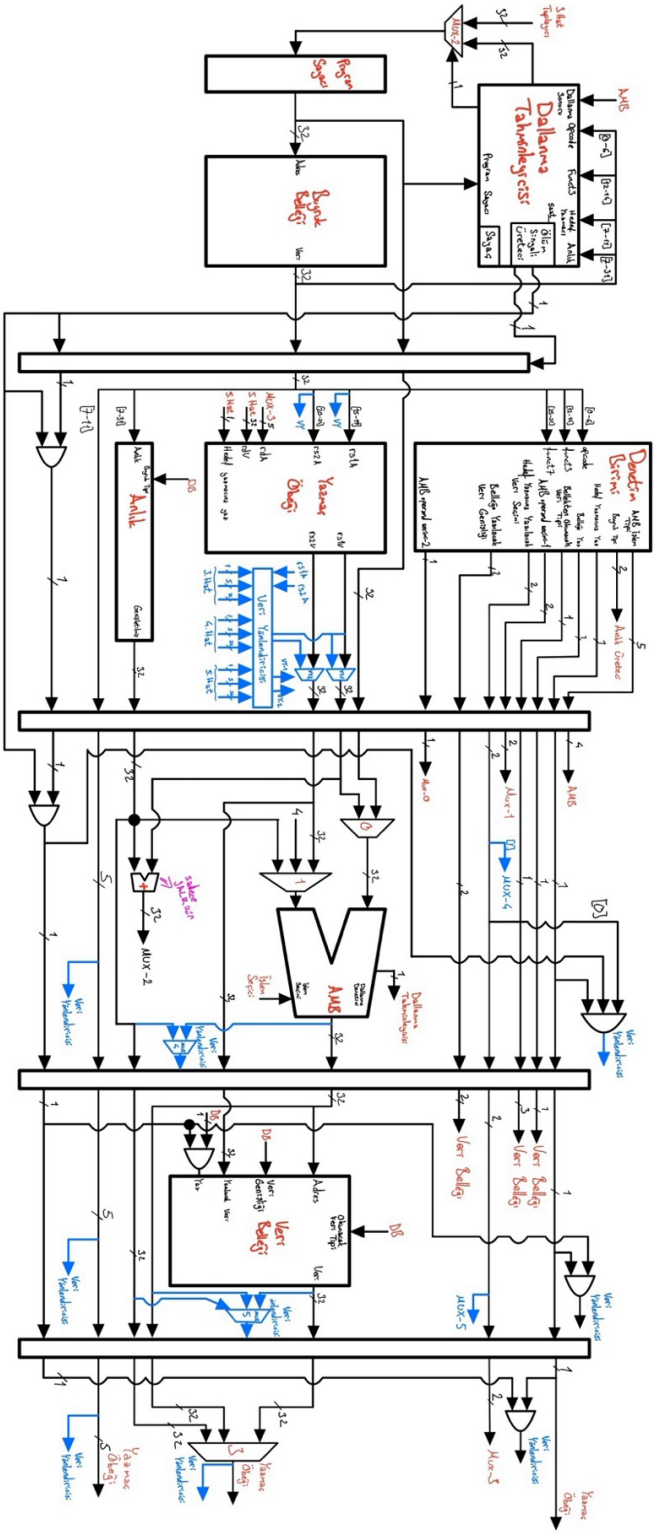
Bu buyruklar ile koşulsuz atlama işlemleri gerçekleştirilir. Genel biçimleri şöyledir:

→ (Buyruk Adı) rd, imm

Bu tipte sadece 1 adet buyruk vardır:

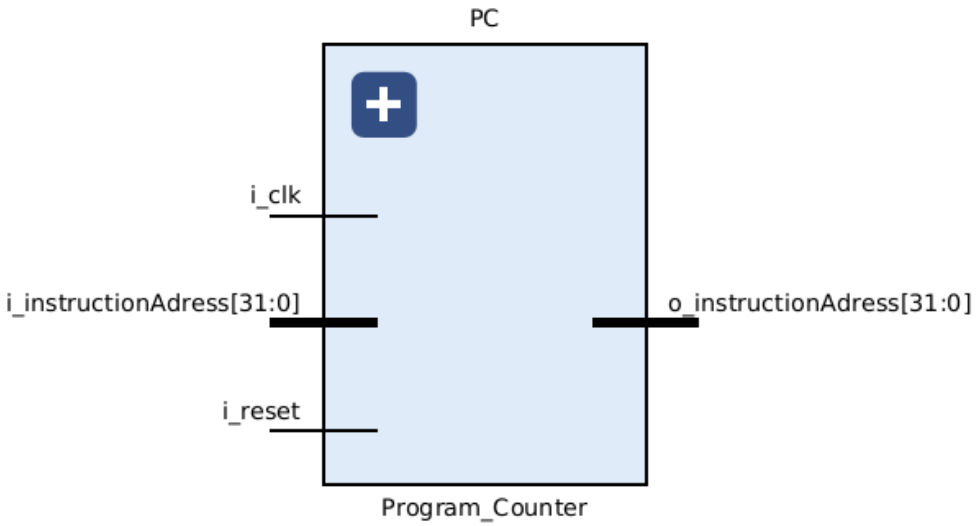
JAL: Jump and Link
→ Program sayacının bir sonraki değeri eski konuma geri dönme imkanı için hedef yazmacına yazılır. Devamında program sayacı ile anlık değer toplanır. Bu toplam da proram sayacına yüklenerek istenilen adrese atlanmış olunur.
→ JAL x22, FFAAD

BÖLÜM 5. BİRİMLER



Şekil 5.1 İşlemcinin Genel Tasarımı

5.1. Program Sayacı



Şekil 5.2. Program sayacı

Girişler:

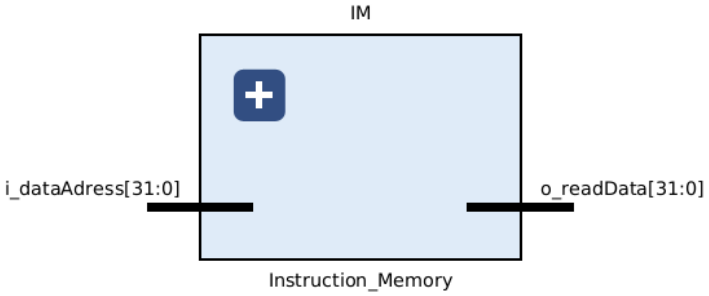
- Saat Sinyali (i_clk – 1 bit)
- Buyruk Adresi Girişi (i_instructionAdress – 32 bit)
- Sıfırlama (i_reset – 1 bit)

Çıkışlar:

- Buyruk Adresi Çıkış (o_ instructionAdress – 32 bit)

Buyruk adresini depolayan bir flip-flop dizisidir. Saat darbesi alınca girişi çıkışa aynı şekilde aktarır. Ayrıca sıfırlama ile içindeki değer kolaylıkla sıfırlanır.

5.2. Buyruk Belleği



Şekil 5.3. Buyruk belleği

Girişler:

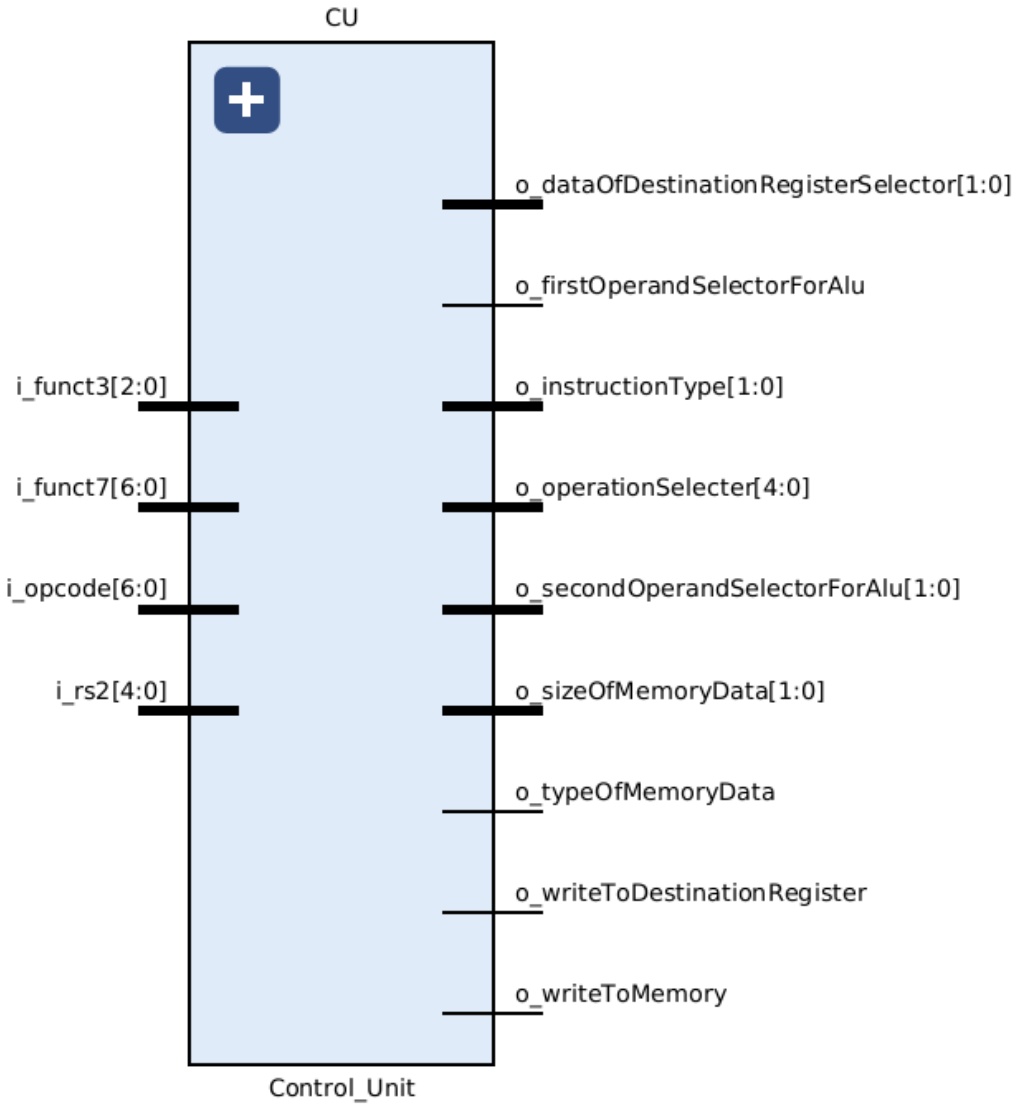
- Buyruk Adresi (i_dataAdress – 32 bit)

Çıkışlar:

- Buyruk (o_readData – 32 bit)

Buyrukların depolandığı bellektir. 1KiB alana sahiptir. Bir saat sinyali girişi yoktur. Kendisine adres verildiğinde buyruğu anında çıkışa aktarır.

5.3. Denetim Birimi



Şekil 5.4. Denetim birimi

Girişler:

- Funct3 (i_funct3 – 3 bit)
- Funct7 (i_funct7 – 7 bit)
- Opcode (i_opcode – 6 bit)
- 2.Kaynak Yazmacı (i_rs2 – 5 bit)

Çıkışlar:

- Hedef Yazmacın Verisinin Seçicisi (o_destinationRegisterDataSelector – 2 bit)
- AMB İşlem Tipi (o_AluOperationType – 5 bit)
- Buyruk Tipi (o_instructionType – 2 bit)
- Hedef Yazmacına Yaz (o_writeDestinationRegister – 1 bit)
- Belleğe Yaz (o_writeMemory – 1 bit)
- Bellek Verisi Tipi (o_memoryDataType – 1 bit)
- AMB Birinci İşlenen Seçicisi (o_AluFirstOperandSelector – 2 bit)
- AMB İkinci İşlenen Seçicisi (o_AluSecondOperandSelector – 1 bit)
- Bellek Verisi Genişliği (o_memoryDataSize – 2 bit)

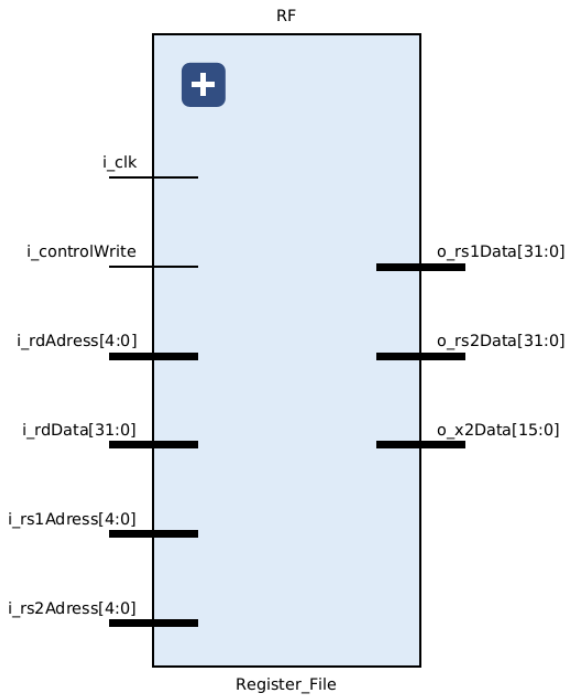
Buyruğun opcode, funct3 ve funct7 kısımlarına bakarak o buyruk için gereken denetim sinyallerini bir saat sinyali olmadan üretir. Her bir çıkış bir boru hattındaki bir şeyi denetler. Amaçları şu şekildedir:

- Hedef Yazmacın Verisinin Seçicisi: Temel görevi 5'inci aşamadaki MUX-3'e bağlanarak hedef yazmacına yazılmak üzere uygun verinin seçimini ayarlar. Bunun yanında 3'üncü, 4'üncü ve 5'inci aşamalardan veri yönlendirmesi yapılırken de kullanılır. Ayrıca 3'üncü aşamdayken veri yönlendirme onayı sinyalinin oluşumunda görev aldığı unutulmamalıdır.

- AMB İşlem Tipi: AMB'de yapılacak işlemi belirtir. İşlemler şöyledir:
    - Toplama
    - Toplama (JALR)

- Buyruk Tipi: RISV-V'de anlıklar buyruk tiplerine göre farklı şekillerde buyruklara dağıtılmıştır. Bu anlıkları düzgün şekilde genişletilmesi için buyruğunun tipinin bilinmesi gerekir. Bu ihtiyaçtan 2'nci aşamadaki Anlık Genişletici birimine buyruğun tipini söyler.
- Hedef Yazmacına Yaz: Bazı buyruklar hedef yazmacına veri yazarlar. Bunun kontrolü bu sinyal aracılığıyla yapılır.
- Belleğe Yaz: Sadece belleğe yazma buyrukları için vardır. Belleğe veri yazılmasını sağlar.
- Bellek Verisi Tipi: Bellekten okunacak verinin işaretli mi veya işaretsiz mi olarak okunacağını belirtir.
- AMB Birinci İşlenen Seçicisi: AMB'nin birinci işlenenin program sayacı mı yoksa birinci kaynak yazmacı mı olacağını belirtir.
- AMB İkinci İşlenen Seçicisi: AMB'nin ikinci işlenenin anlık mı, 4 sayısı mı yoksa ikinci kaynak yazmacı mı olacağını belirtir.
- Bellek Verisi Genişliği: Bellekten okunacak ya da belleğe yazılacak verinin 8 bit mi, 16 bit mi yoksa 32 bit mi olacağını belirtir.

5.4. Yazmaç Öbeği



Şekil 5.5. Yazmaç öbeği

Girişler:

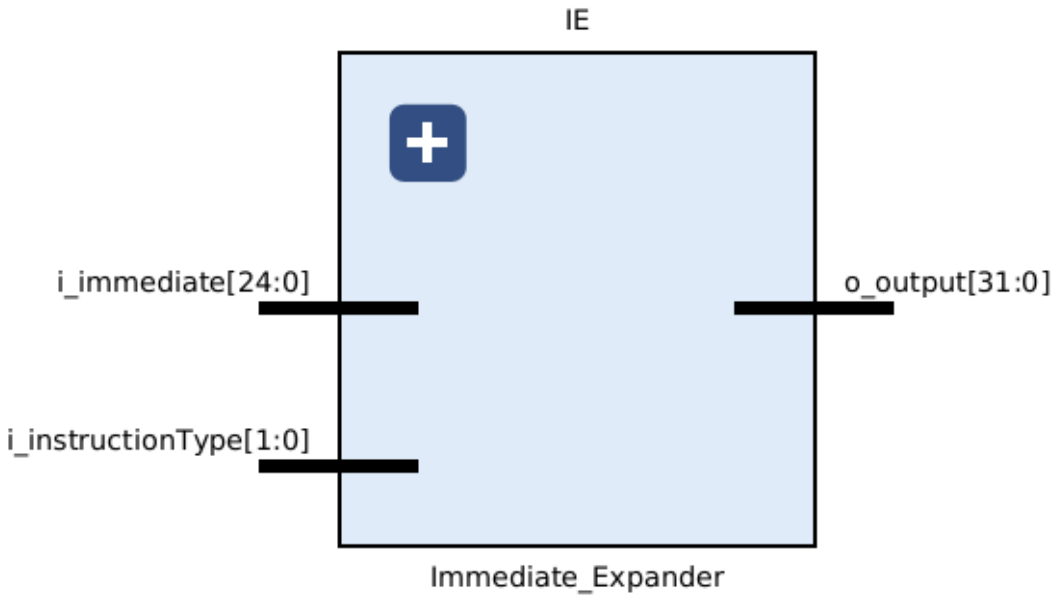
- Saat Sinyali (i_clk – 1 bit)
- 1. Kaynak Yazmacın Adresi (i_rs1Adress – 5 bit)
- 2. Kaynak Yazmacın Adresi (i_rs2Adress – 5 bit)
- Hedef Yazmacın Adresi (i_rdAdress – 5 bit)
- Hedef Yazmacın Verisi (i_rdData – 5 bit)
- Hedef Yazmacına Yaz (o_writeDestinationRegister - 1 bit)

Çıkışlar:

- 1. Kaynak Yazmacın Verisi (o_rs1Data – 32 bit)
- 2. Kaynak Yazmacın Verisi (o_rs2Data – 32 bit)
- 3. Yazmacın İlk 16 Biti (o_x2Data – 16 bit)

Yazmaçların bir arada tutulduğu ve saat sinyali olmayan bir birimdir. İçinde 32 tane 32 bitlik yazmaç bulunur. Sıfırıncı yazmacın değeri her zaman 0'dır. Asla değiştirilemez. 2 farklı yazmacın verisini aynı anda dışarı aktarabilir. Sadece 1 adet yazmacın verisini değiştirebilir.

5.5. Anlık Genişleticisi



Şekil 5.6 Anlık Genişleticisi

Girişler:

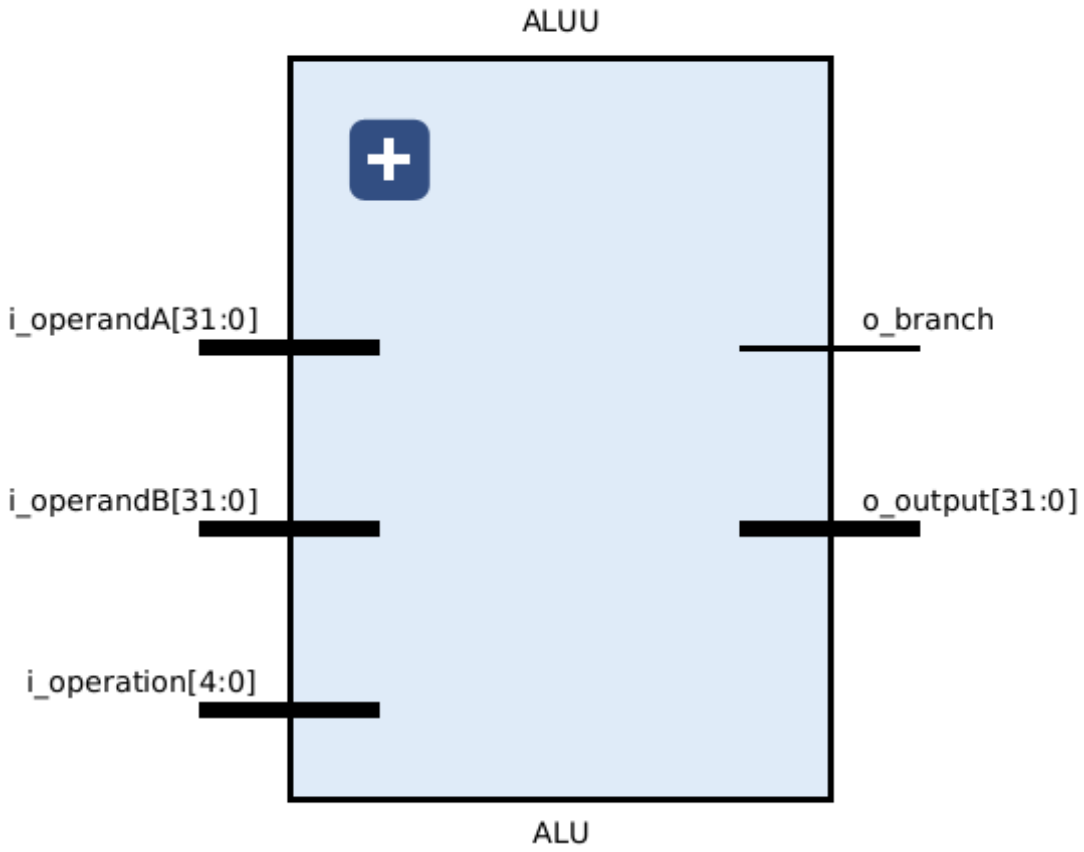
- Anlık (i_immediate – 25 bit)
- Buyruk Tipi (i_instructionType – 3 bit)

Çıkışlar:

- Genişletilmiş Anlık (o_output – 32 bit)

İlk, buyruğun [7-31] bitlerini alacak. Çünkü farklı tipteki buyrukların anlıkları bu bitlere dağılmış durumdadır. İkinci girişe ise buyruğun tipi verilecektir. Anlık değer kullanan 5 buyruk tipi vardır. Bunlardan "J" ve "B" tipi olanlar dallanma öngörücüsü içeresinde anlık değerleri oluşturulacak. Geri kalan 3 buyruğun ise burada oluşturulacak. Buyruk tipine uygun olacak şekilde anlığın değerler genişletilir. Bütün anlık değerler işaretli sayı olarak kabul edilerek genişletilir.

5.6. Aritmetik Mantık Birimi



Şekil 5.7. Aritmetik mantık birimi

Girişler:

- Birinci İşlenen (i_operandA – 32 bit)
- İkinci İşlenen (i_operandB – 32 bit)
- İşlem (i_operation – 5 bit)

Çıkışlar:

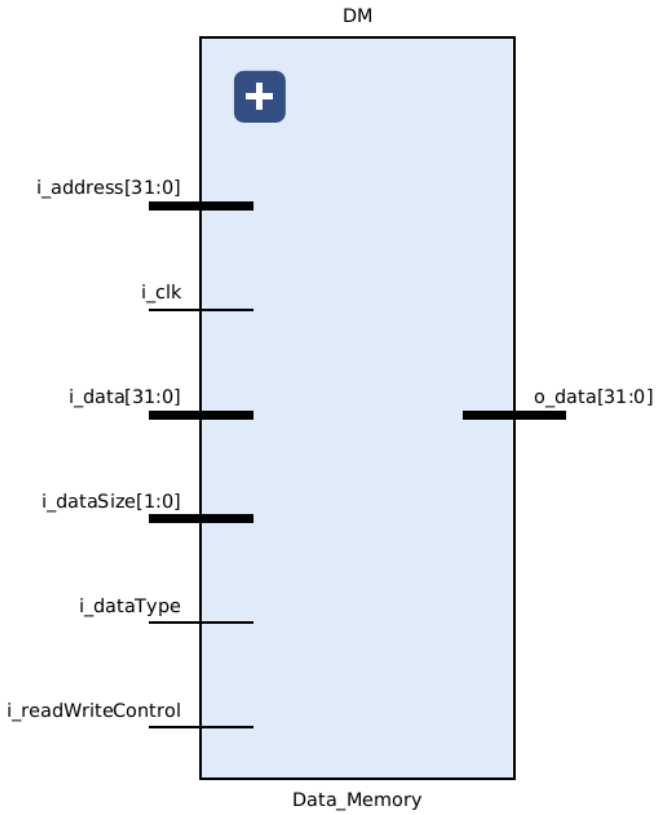
- Sonuç (o_result – 32 bit)
- Dallan (o_branch – 1 bit)

Yaptığı İşlemler:

Toplama
Toplama (JALR)
Çıkarma
VE
VEYA
ÖZEL VEYA
Sola Kaydırma
Sağa Kaydırma
Sağa Kaydırma (Aritmetik)
Küçük Mü (İşaretli)
Küçük Mü (İşaretsiz)
Büyük/Eşit Mi (İşaretli)
Büyük/Eşit Mi (İşaretsiz)
Eşit Mi
Eşit Değil Mi
Hamming Uzaklığı
PKG İşlemi
RVRS İşlemi
SLADD İşlemi
CNTZ İşlemi
CNTP İşlemi

JALR buyruğu için AMB, yazmaçtaki değer ile anlığı topladıktan sonra RISC-V'in isterlerinden dolayı en önemsiz biti sıfır yapması gerekir. Bu işlem AMB yerine özel bir toplayıcı ile yapılmıştır.

5.7. Veri Belleği



Şekil 5.8 Veri Belleği

Girişler:

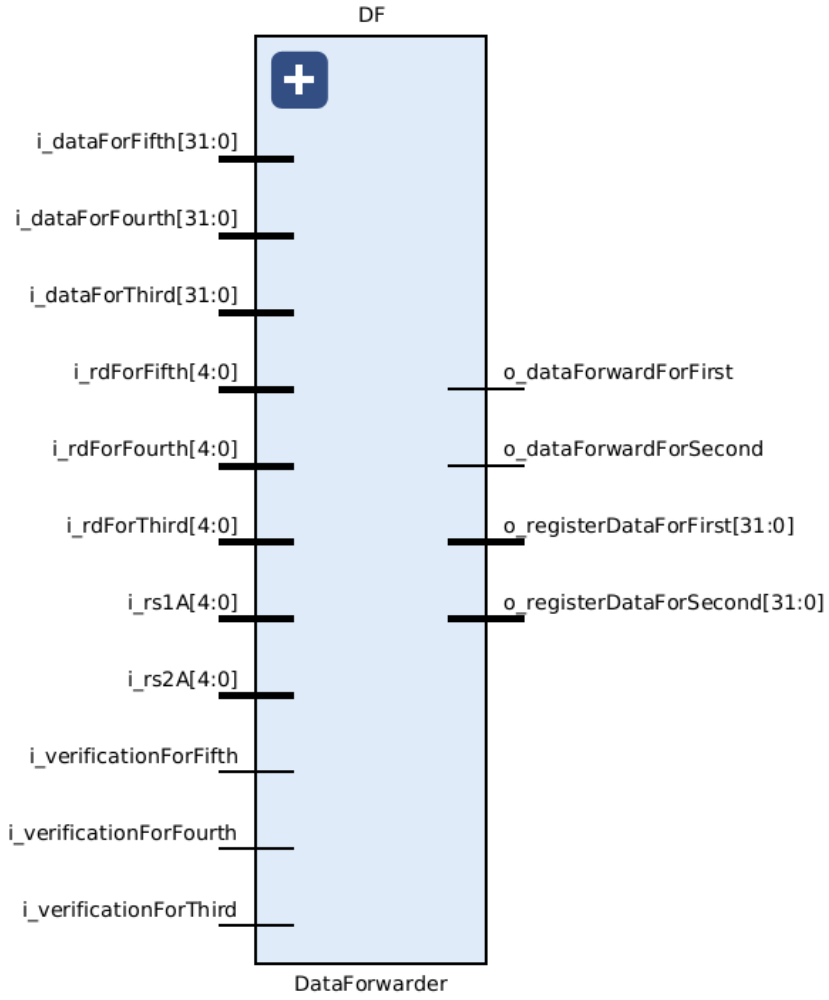
- Saat Sinyali (i_clk – 1 bit)
- Adres (i_adress – 32 bit)
- Veri Genişliği (i_dataSize – 2 bit)
- Veri Tipi (i_dataType – 1 bit)
- Veri Yaz (i_writeData – 1 bit)
- Yazılacak Veri (i_data – 32 bit)

Çıkışlar:

- Okunan Veri (o_data – 32 bit)

Verilerin tutulduğu ve 1 KiB alana sahip olan bellektir.

5.8. Veri Yönlendiricisi



Şekil 5.9 Veri Yönlendiricisi

Girişler:

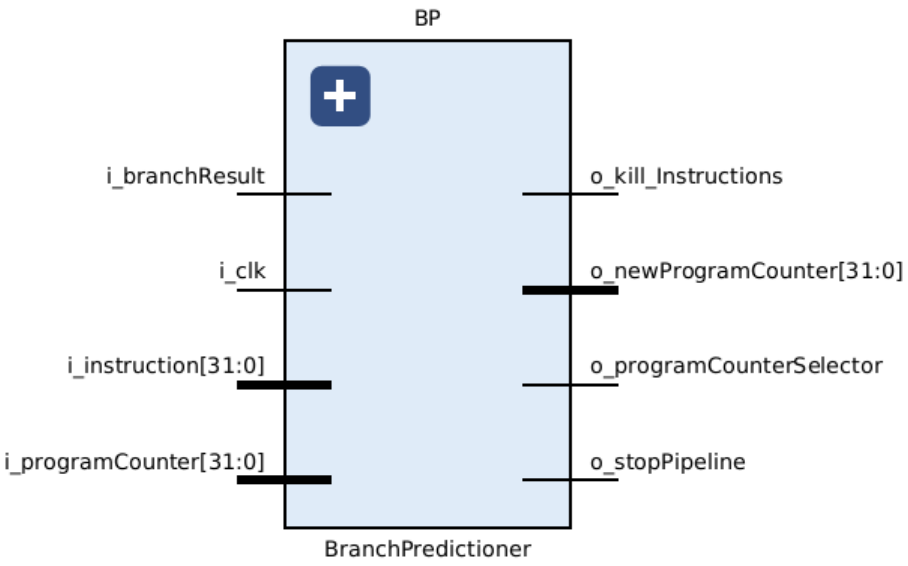
- 1.Kaynak Yazmacın Adresi (i_rs1A – 5 bit)
- 2.Kaynak Yazmacın Adresi (i_rs2B – 5 bit)
- 3.Aşama Onay (i_verificationForThird – 1 bit)
- 3.Aşama Hedef Yazmacın Adresi (i_rdForThird – 5 bit)
- 3.Aşama Veri (i_dataForThird – 32 bit)
- 4.Aşama Onay (i_verificationForFourth – 1 bit)
- 4.Aşama Hedef Yazmacın Adresi (i_rdForFourth – 5 bit)
- 4.Aşama Veri (i_dataForFourth – 32 bit)
- 5.Aşama Onay (i_verificationForFifth – 1 bit)

- 5.Aşama Hedef Yazmacın Adresi (i_rdForFifth– 5 bit)
- 5.Aşama Veri (i_dataForFifth– 32 bit)

Çıkışlar:

- 1.Yazmaç İçin Veri Yönlendiricisi (o_dataForwardForFirst – 1 bit)
- 2.Yazmaç İçin Veri Yönlendiricisi (o_dataForwardForSecond – 1 bit)
- 1.Yazmaç Verisi (o_registerDataForFirst – 1 bit)
- 2.Yazmaç Verisi (o_registerDataForSecond – 1 bit)

5.9. Dallanma Öngörücüsü



Şekil 5.10 Dallanma Öngörücüsü

Girişler:

- Saat Sinyali (i_clk – 1 bit)
- Program Sayacı (i_programCounter – 32 bit)
- Buyruk (i_instruction – 32 bit)
- Dallanma Sonucu (i_predictionResult – 1 bit)

Çıkışlar:

- Yeni Program Sayacı (o_newProgramCounter – 32 bit)
- Program Sayacı Seçicisi (o_programCounterSelector – 1 bit)
- Buyrukları Öldür (o_kill_Instructions – 1 bit)
- Boru Hattını Durdur (o_stopPipeline – 1 bit)

BÖLÜM 6. Uygulamalar

Geliştirilen işlemcinin yeteneklerini gösterebilmek için 3 farklı uygulama yapılmıştır. Bu uygulamalar genel olarak işlemcideki buyrukları gösterir. İşlemcinin potansiyeli özünde bu uygulamaların çok daha ötesindedir. Bütün uygulamalar, simülasyon aracılığıyla test edilmiştir.

6.1. Temel Aritmetik ve Mantıksal İşlemler

Assembly Kodu:

```
addi x1,x0,6
add x1,x1,x1
add x1,x1,x1
add x1,x1,x1
addi x2,x0,2

add x3,x1,x2
sub x3,x1,x2
and x3,x1,x2
or x3,x1,x2
xor x3,x1,x2
sll x3,x1,x2
srl x3,x1,x2
slt x3,x1,x2

addi x3,x1,4
andi x3,x1,4
ori x3,x1,4
xori x3,x1,4
slli x3,x1,4
```

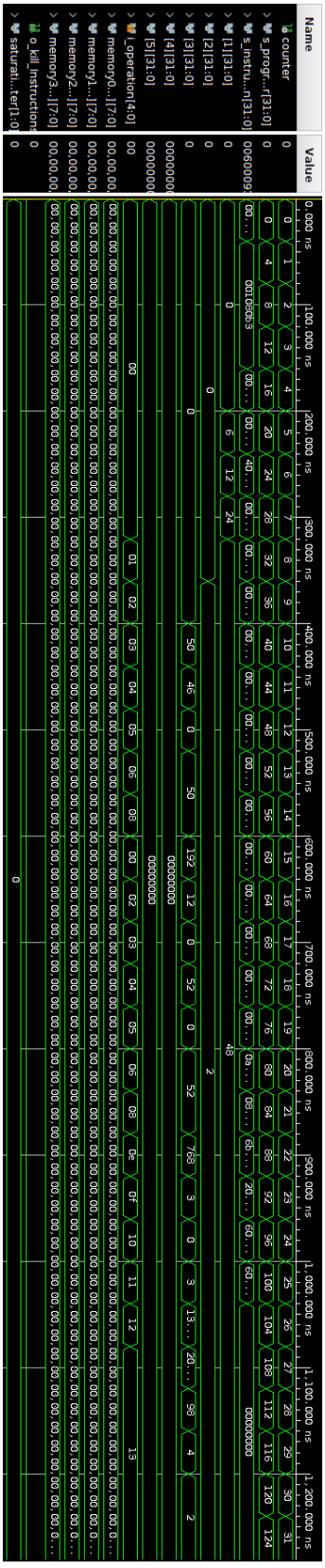
```
srli x3,x1,4
slti x3,x1,4

hmdst x3,x1,x2
pkg x3,x1,x2
rvrs x3,x1
sladd x3,x1,x2
cntz x3,x1
cntp x3,x1
```

Bu uygulamada ilk önce x1 yazmacına anlık değer olarak “48” yüklenmek isteniyor. Normalde bu anlık değer tek seferde yüklenebilir fakat sistemdeki bulunan veri yönlendirmesi mekanizmasını test etmek için ilk önce yazmaca 6 yükleniyor sonra 3 defa kendisiyle toplama işlemine tabi tutulur. Sonrasında ise x2 yazmacına anlık “2” değeri yüklenir.

Devamındaki 8 buyruk x1 ile x2 yazmacı üzerinde aritmetik ve mantıksal işlemler yaparlar. 7 buyruk boyuncada x1 yazmacındaki veri ile anlık “4” değeri ile bazı işlemlere tabi tutulur. Son 6 buyruk ile de kriptografi işlemlerini hızlandıran buyruklar kullanılmıştır. Bütün sonuçlar x3 yazmacına yazıldırılmıştır.

Simülasyon Sonucu:



Şekil 6.1 Simülasyon Sonucu - 1

6.2. Bellek İşlemleri

Assembly Kodu:

```
lui x3,0x10101
addi x3,x3,0x010
add x5,x3,x0
sb x3,0(x0)
srli x3,x3,8
sb x3,1(x0)
srli x3,x3,8
sb x3,2(x0)
srli x3,x3,8
sb x3,3(x0)
lw x4,0(x0)
slt x3,x4,x5

lui x3,0xA0A0A
addi x3,x3,0x0A0
add x5,x3,x0
sh x3,4(x0)
srli x3,x3,16
sh x3,6(x0)
lw x4,0(x0)
slt x3,x4,x5

lui x3,0x11111
addi x3,x3,0x111
add x5,x3,x0
sw x3,8(x0)
lw x4,0(x0)
slt x3,x4,x5
```

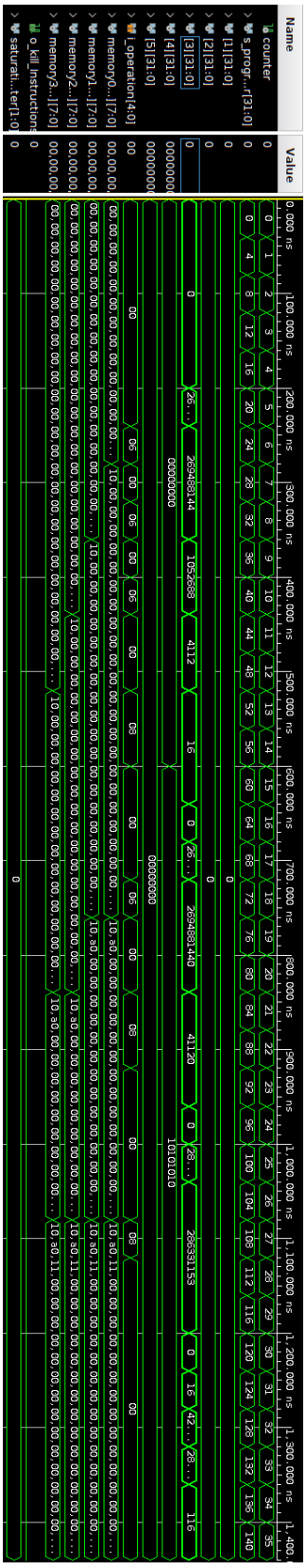
```
lb x3,0(x0)
lh x3,4(x0)
lw x3,8(x0)
auipc x3,0
```

Bu uygulamada işlemcinin bellek okuma ve yazma yetenekleri gözlemlenir.

Burada aslında sırasıyla “0x10101010”, “0xA0A0A0A0” ve “0x11111111” değerleri belleğe kaydedilmek isteniyor. Sayıların her biri 32 bitlik birer sayı olduğu için tek seferde yüklenemezler. Bunun için ilk önce 20 bitlik kısımları “LUI” aracılığıyla, 12 bitlik kısmı ise “ADDI” ile x3 yazmacına yüklenir. Yükleme işlemi bittikten bu değerlerin bir yedeği x5 yazmacına aktarılır. Sonra ilk sayı belleğe bayt bayt yazılır, ikinci sayı 2 bayt 2 bayt, son sayı ise tek seferde 4 bayt olarak yazılır. Devamında belleğe yazılan bu sayılar 4 bayt olacak şekilde okunarak x4 yazmacına yazılır. Bu okunan değer ile x3 yazmacındaki değerler “SLT” buyruğu ile aynı olup olmadığı sonucu yine x3 yazmacına yazılarak gözlemlenir. Bunlardan sonra 1. sayının ilk baytı, 2. sayının iki baytı ve 3. sayının 4 baytı x3 yazmacına sırayla yazılır. Bununla bellekten farklı boyutlarda veri okuyabilme yeteneğini gösterilmiştir. En sonda ise son buyruğun hangi adreste olduğu x3 yazmacına yazılır.

Burada veri yönlendirmesinin bariz bir şekilde yardımını görüyoruz. Şöyle ki boru hattı dolduktan sonra her saat darbesiyle bir buyruk işlenmiş oluyor. Eğer veri yönlendirmesi olmasaydı boru hattında beklemelere neden olacak ve çok daha yavaş çalışacaktı.

Simülasyon Sonucu:



Şekil 6.2 Simülasyon Sonucu - 2

6.3. Döngü ve Dallanma Tahmini

Assembly Kodu:

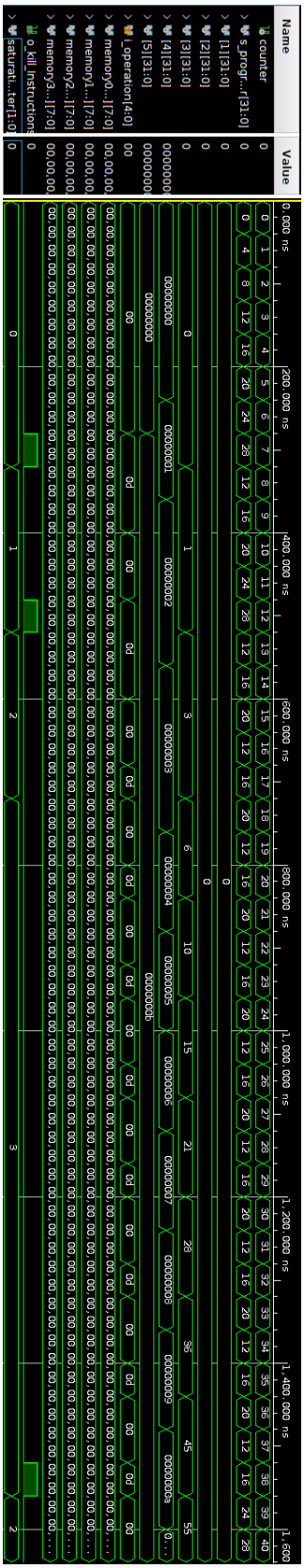
```
x"93", x"01", x"00", x"00", -- addi x3,x0,0
x"13", x"02", x"10", x"00", -- addi x4,x0,1
x"93", x"02", x"b0", x"00", -- addi x5,x0,11

x"b3", x"81", x"41", x"00", -- add x3,x3,x4
x"13", x"02", x"12", x"00", -- addi x4,x4,1
x"e3", x"1c", x"52", x"fe"  -- bne x4,x5,loop
```

Burada 1'den 10'a kadar olan sayıları toplayan bir program yazılmıştır. Bunun için de döngü kullanılmıştır. x3, x4 ve x5 yazmaçlarına sırasıyla 0, 1 ve 11 değerleri yazılmıştır. Burada x3 yazmacı toplam değer, x4 yazmacı sayaç ve x5 yazmacı ise sayacın sınırı olarak kullanılmıştır. Bundan sonra sürekli olarak x3 ve x4 yazmaçlarındaki değerler toplanır. Burada döngü oluşturmak için "BNE" buyruğu kullanılmıştır. Sayaç 11 değerini görene kadar bu döngü devam ettirilir.

Burada işlemcinin dallanma öngörüsü sistemi kendisini ön plana çıkarır. Geliştirilen işlemcideki dallanma öngörüsü sistemi başlangıçta "kararlı - dallanma" olarak ayarlıdır. Bu koddaki "BNE" buyruğu normalde dallanılması gerekirken işlemci dallanmayacaktır. Sonrasında yanlış tahmin olduğu ortaya çıkınca o süre zarfında yapılan işlemler iptal edilir ve dallanılması gereken yeri dallanıp oradan devam edilir. Sistem yanıldığından dolayı "kararlı – dallanma" durumundan "kararsız – dallanma" durumuna geçer. Bu durumda hala sistem "dallanma" şeklinde karar verecektir. Bundan sonra sistem yine atlanılması gereken yerden atlamadığı için yine yanılgıya düşer. Fakat bu sefer "kararsız – dallan" durumuna geçtiği için artık kararı "dallan" olarak değişmiştir. Bundan sonra döngünün sonuna kadar işlemci "dallan" olarak tahmin yapacaktır. En sonda sayaç, 11'e eşit olunca sistemin artık dallanmaması gerekecektir fakat sistem "dallan" olarak karar verecektir. Bundan dolayı da bir kez daha sistem hataya düşmüş olur.

Simülasyon Sonucu:



Şekil 6.3 Simülasyon Sonucu - 3

BÖLÜM 7. SONUÇ

Projede bilgisayar mimarisi ve FPGA üzerine araştırmalar yapılmıştır. Araştırmalar doğrultusunda bir RISC-V RV32-I buyruk kümesine sahip bir işlemciye karar kılınmıştır. İşlemcide boru hattı, dallanma öngörüsü ve veri yönlendirme yöntemlerinin kullanılmasına karar verilmiştir.

Ayrıca işlemcinin VHDL ile yazılıp FPGA gerçeklenmesine karar verilmiştir. Bunlar doğrultusnda 5 aşamalı bir boru hattına sahip işlemci dizayn edilmiştir. İşlemcide veri bağımlılıklarını çözmek için veri yönlendirmesi de yapılmıştır. RV32-I'daki 40 buyruktan 37'si bu tasarıma dahil edilmiştir. Bunun yanında kriptografi işlemleri için özel buyruklar da eklemiştir. İşlemci birimleri VHDL ile yazılmıştır.

İşlemcide şöyle bir sorun bulunmaktadır: Bellekten veri yükleme buyruğundan hemen sonra verinin yüklendiği yazmaç kullanılmaya çalışılınca bazı durumlarda hatalı sonuçlar verebiliyor. Bir de işlemcinin simülasyon ile az önceki hata dışında bir hata ile karşılaşılmamasına rağmen FPGA üzerindeki gerçeklemesinde hatalı sonuçlar vermektedir.

Geliştirilen işlemcinin daha iyi hale getirmek için birçok farklı şey yapılabilir. Bunları şu şekilde sıralayabiliriz:

- Yukarıda belirtilen 2 problem çözülebilir.
- Buyrukların iletilebilmesi için UART arayüzü konabilir.
- Ön Bellek eklenebilir.
- Daha iyi tahmin yapan bir dallanma öngörücüsü yapılabilir.
- Çoklu çekirdek şeklinde tasarlanabilir.
- RISC-V'in diğer buyruk uzantıları eklenebilir.

KAYNAKÇA

[1] «Teknofest Çip Tasarım Yarışması Şartnamesi,» 2022.

[2] [Çevrimiçi]. Available: https://www.terasic.com.tw/cgi-bin/page/archive.pl?Language=English&No=502. [Erişildi: 31 05 2024].

[3] «RiscV Logo,» [Çevrimiçi]. Available: https://riscv.org/.

[4] «risc hakkında,» [Çevrimiçi]. Available: https://riscv.org/about/history/.

[5] [Çevrimiçi]. Available: https://standards.ieee.org/ieee/1076/5179/.

[6] [Çevrimiçi]. Available: https://ieeexplore.ieee.org/document/8938196.

[7] J. E. SMITH, «A STUDY OF BRANCH PREDICTION STRATEGIES,» 1981.

[8] «Pipelining,» [Çevrimiçi]. Available: https://lishixuan001.com/posts/3920/. [Erişildi: 31 05 2024].

[9] C. R. P. S. Claire Burguière, «A Case for Static Branch Prediction in Real-Time Systems,» 2005.

[10] [Çevrimiçi]. Available: https://www.intel.com/content/www/us/en/products/details/fpga/development-tools/quartus-prime.html..

[11] «qwe,» [Çevrimiçi]. Available: https://faculty.cc.gatech.edu/~hyesoon/fall11/prog1.html.