T.C.
BEYKENT ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ
BİLGİSAYAR MÜHENDİSLİĞİ ANABİLİM DALI
BİLİGİSAYAR MÜHENDİSLİĞİ BİLİM DALI

MİKROİŞLEMCİ MİMARİLERİNDE BOOLEAN CEBRİ ÜZERİNE GELİŞTİRMELER VE KARŞILAŞTIRMA İŞLEMLERİ

Yüksek Lisans Tezi

Tezi Hazırlayan:
Abubekir ŞAKAR

İstanbul, 2018

T.C.
BEYKENT ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ
BİLGİSAYAR MÜHENDİSLİĞİ ANABİLİM DALI
BİLİGİSAYAR MÜHENDİSLİĞİ BİLİM DALI

MİKROİŞLEMCİ MİMARİLERİNDE BOOLEAN CEBRİ ÜZERİNE GELİŞTİRMELER VE KARŞILAŞTIRMA İŞLEMLERİ

Yüksek Lisans Tezi

Tezi Hazırlayan:
Abubekir ŞAKAR

Öğrenci No:

140820031

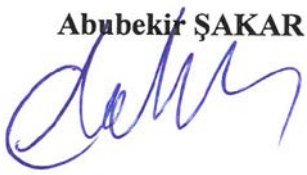
Danışman:
Yrd. Doç. Dr. Ediz ŞAYKOL

İstanbul, 2018

YEMİN METNİ

Yüksek Lisans Tezi olarak sunduğum “Mikroişlemci Mimarilerinde Boolean Cebri Üzerine Geliştirmeler ve Karşılaştırma İşlemleri” başlıklı bu çalışmanın, bilimsel ahlak ve geleneklere uygun şekilde tarafımdan yazıldığını, yararlandığım eserlerin tamamının kaynaklarda gösterildiğini ve çalışmamın içinde kullanıldıkları her yerde bunlara atıf yapıldığını belirtir ve bunu onurumla doğrularım. 08/01/2018

Abubekir ŞAKAR



T.C.
BEYKENT ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

YÜKSEK LİSANS TEZ SAVUNMA SINAVI SONUÇ TUTANAĞI

Beykent Üniversitesi
Fen Bilimleri Enstitüsü Müdürlüğü'ne,

Aşağıda tez adı belirtilen yüksek lisans öğrencisi 140820031 no'lu ABUBEKİR SAKAR 'in 15./04./2015 tarihinde yapılan tez savunma sınavı[1] sonucunda 45 dakika süreyle sunduğu ve savunduğu tezi hakkında[2] oybirliğiyle, KABUL kararı verilmiştir.

Bilgilerinize saygılarımızla arz ederiz.

Anabilim Dalı : Bilgisayar Mühendisliği
Programı : Bilgisayar Mühendisliği
Tez Başlığı[3] : Mikroişlemci Mimarisinde Bodean Cebri Üzerine Gelişmeler ve Karşılaştırma İşlemleri

---

Tez Sınav Jürisi	Öğretim Üyesi	İmza
Danışman	: Yrd. Doç. Dr. Ediz ŞAKOĞLU	
Üye	: Doç. Dr. Gökhan SİLAHTAROĞLU	
Üye	: Yrd. Doç. Dr. Turhan KARAGÜLER	

---

[1] Jüri üyeleri söz konusu tezin kendilerine teslim edildiği tarihten itibaren en geç bir ay içinde toplanarak öğrenciyi tez savunma sınavına alır. Belirlenen günde yapılamayan jüri toplantısı, katılanların hazırladığı bir tutanakla enstitü yönetimine bildirilir. Bu durumda jüri en geç onbeş gün içinde toplanarak adayı tez savunma sınavına alır. Tez savunma sınav süresi en az 45 dakikadır. Yüksek lisans tez savunma sınavı, tez çalışmasının sunulması ve bunu izleyen soru-yanıt bölümlerinden oluşur ve dinleyiciye açıktır. (Beykent Lisansüstü eğitim ve Öğretim Yönetmeliği-Madde30-3)

[2] Tez sınavının tamamlanmasından sonra jüri, tez hakkında "kabul", "düzeltme" veya "red" kararı verir. Jüri başkanı, jüri üyelerince imzalanmış sınav tutanağını, tez sınavını izleyen üç gün içinde ilgili enstitü yönetimine teslim eder. Tezi başarısız bulunan öğrencinin Enstitü ile ilişiği kesilir. Tezi hakkında düzeltme kararı verilen öğrenci en geç üç ay içinde gerekli düzeltmeleri yaparak ve yönetmelikte belirtilen usullere uygun olarak tezini aynı jüri önünde yeniden savunur. Bu savunma sınavında da tezi kabul edilmeyen öğrencinin enstitü ile ilişiği kesilir. (Beykent Lisansüstü eğitim ve Öğretim Yönetmeliği-Madde30-4)

[3] İleride doğabilecek aksaklıkların engellenmesi için tezin başlığının yazılması gerekmektedir.

Adı ve Soyadı :Abubekir ŞAKAR

Danışmanı :Yrd. Doç. Dr. Ediz ŞAYKOL

Türü ve Tarihi :Yüksek Lisans, 2018

Alanı :Bilgisayar Mühendisliği

Anahtar Kelimeler : Moore Yasası, mikroişlemci mimarisi, karşılaştırma işlemleri, yüksek seviyeli yazılım dilleri, MIPS

ÖZ

MİKROİŞLEMCİ MİMARİLERİNDE BOOLEAN CEBRİ ÜZERİNE GELİŞTİRMELER VE KARŞILAŞTIRMA İŞLEMLERİ

Mikroişlemci mimarileri 1960'lı yıllardan beri dünya genelinde gelişmektedir. Gelişim ilk yıllardan itibaren ivmelenerek devam etmiştir. Bu gelişimin fonksiyonel açıklaması Moore Yasası olarak izah edilmiş ve her 18 ayda mikroişlemcilerdeki bileşenlerin sayısının iki katına çıkacağı ve maliyetinin aynı kalacağı öngörülmüştür. Bu gelişim ilk yıllardan itibaren beklenenden fazla olmasına rağmen tek çekirdekli bir işlemciyi ele alındığında son on yılda işlemcilerdeki büyüme duraksamıştır. İşlemcilerde frekans artışı bu son dönemde 4Ghz bandı civarında kalıp, gelişmeler işlemcilerin çekirdek sayılarında ve mimari yapılarında olmuştur. Bu yazı kapsamında işlemci mimarilerinde, yüksek seviyeli yazılım dillerinin derleme sonucu performanslarında geliştirme ve eğitim açısından kod okunurluğunu artırmak amacıyla boolean cebrine ait işlemler, karşılaştırma ve dallanma işlemlerinde geliştirmeler planlanacak ve MIPS mimarisi üzerinde gelişimin nasıl olacağı anlatılacaktır.

Name and Surname	:Abubekir ŞAKAR
Supervisor	: Assistant Professor Ediz ŞAYKOL
Degree and Date	:Master, 2018
Major	:Computer Engineering
Keywords	: Moore law, microprocessor architecture, comparsion operations, high level software languages, MIPS architecture

ABSTRACT

DEVELOPMENTS ON BOOLEAN ALGEBRA AND COMPARSION OPERATIONS IN THE MICROPROCESSOR ARCHITECTURES

Microprocessor architects have been developing around the world since the 1960s. The development has continued since the first years by increasing the acceleration. The functional description of this development has been explained as Moore's Law and it is predicted that the cost will be the same as twice the number of components in microprocessors every 18 months. Although this development has been expected to be more than expected from the first years, the growth of the processors has stagnated in the last decade when a single-core processor is considered. The increase in frequency in the processors has been in the 4GHz band in the last period, and the core numbers and architectural structures of the improvement processors. In this paper, processor architects will be developing boolean algebraic operations comparsion and branching operations in order to boost the readability of the high level software languages compiling results in terms of development performance and training also it will be explained how to develop on MIPS architecture.

İÇİNDEKİLER

TABLOLAR LİSTESİ

ŞEKİLLER LİSTESİ

KISALTMALAR

ALU : Arithmetic Logic Unit (Aritmetik Mantık Birimi)

ARM : Acorn RISC Machine

BLU : Bit Logical Unit

BRA : Bit Register Array

BTU : Bit Transfer Unit

CISC : Complex instruction set computer (Karmaşık komut setli bilgisayar)

EX : Execute

FPGA : Field Programmable Gate Array (Alanda Programlanabilir Kapı Dizileri)

HDL : Hardware Description Language (Donanım Tanımlama Dili)

ID : Instruction Decoder

IF : Instruction Fetch

MEM : Memory

MIPS : Million instructions per second (saniyede milyon işlem)

RA : Register Array – Register File

RCU : Register Comparsion Unit

RISC : Reduced instruction set computing (İndirgenmiş komut takımıyla hesaplama)

WB : Writeback

1. GİRİŞ

Bu yazıda mikroişlemci mimarileri üzerinde boolean ve dallanma işlemleri üzerine çalışmalar yapılacaktır. Bu çalışmalarda MIPS mimarisine uygun olarak dört yeni birim tasarlanacak, bu birimlerin komut setine ve MIPS çekirdeğine eklenebilmesi üzerine çeşitli çalışmalar yürütülecektir. Buradaki temel amaç ise karşılaştırma işlemlerinin derlenmesinde daha kaliteli ve okunur sonuçlar elde etmek olacaktır. Eklenecek birimler Bit Register Array (BRA), Bit Logical Unit (BLU), Bit Transfer Unit (BTU) ve Register Comparsion Unit (RCU) olacaktır.

BRA birimi çeşitli işlemlerden sonra elde edilecek olan bir bitlik sonuçların kaydedileceği alandır. Bu birimi MIPS üzerindeki register yapısına benzetbiliriz.

BLU birimi bitler arasında işlem yapmayı sağlayan birimdir. Bu birimle lojik işlemler BRA biriminden alınarak bitler arasında yapılacak ve sonuçları BRA birimine kaydedilecektir. Buradaki sonuçlara göre dallanma işlemleri yapılabilir olacaktır.

BTU birimi ise MIPS üzerinde bulunan 32 bit boyutundaki bir register üzerinden herhangi bir bit değerini almayı veya bir register üzerine BLU üzerinde bulunan herhangi bir biti kaydetmeyi sağlayacaktır.

Son olarak RCU birimi herhangi iki register üzerindeki veriyi karşılaştıracak ve sonuç olarak oluşan 1 bitlik veriyi BRA üzerine kaydetmeyi sağlayacaktır. Bu karşılaştırma işlemleri büyüktür küçüktür eşittir gibi farklı işlemleri olacaktır.

Çalışmanın temel hedefi herhangi bir işlemci mimarisinde bu ve benzeri geliştirmeler yapıldığında mimari için yazılmış veya yüksek seviyeli bir dilden derlenmiş olan assembly komutlarının okunurluğunu arttırmak ve derleme sonucu çıkan kodun performans açısından iyileşmesi sağlamak olacaktır. Çalışma sonucunda MIPS mimarisine tam bir ekleme yapılmayacak fakat ekleme yapılması düşünülürse nasıl bir yol izleneceği açıklanacaktır.

2. KARŞILAŞTIRMA İŞLEMLERİNİN ÖNEMİ

Karşılaştırma işlemleri programlama dillerinin olmazsa olmaz öğelerindendir. Çünkü programlama dillerinde yukarıdan aşağıya doğru giden doğrusal programlama akışını farklı bölgelere çekmeyi fonksiyonlar, kesmeler ve karşılaştırma işlemleri sağlar. Örnek olarak bir sayıcı programında sayıcının her değişiminde bir döngü yapılmalı ve tekrar değişim için programın doğrusal aşağı akışı durdurulup üst satırlara tekrar çıkarılması gerekmektedir. Bu sayıcı uygulamasında anlatılan bölümün her hangi bir yerinde karşılaştırma işlemi gerekmemesine rağmen sayıcının sonsuza kadar hareket etmemesini karşılaştırma işlemi sağlayacaktır. Bu sayede istenilen değere geldiği takdirde döngü durdurulacak ve işlem başarıyla tamamlanmış olacaktır. Herhangi bir döngünün durdurulması kesme işlemi ile yapılabilse de genellikle kesme işlemleri için uygun değildir. Sayıcı uygulamasına geri dönersek sayıcı uygulamasında kesme ile durdurma yapmak yazılımcıya sadece kesme arasında geçen sayım bilgisini verir. Döngü işlemi kullanmak yerine recursive bir fonksiyon tanımlanabilse de, Bu durumda da recursive ifadenin durdurulma zamanını bir karşılaştırma işlemi belirlemek zorunda olacaktır.

Karşılaştırma işlemleri akla gelince ilk planda “if, else” ifadelerini anlatmasına rağmen “if, else” ifadeleri sadece karşılaştırma işleminin sonucuna göre faaliyet gösteren birer program kod parçacığıdır. Asıl karşılaştırma işlemi ise bu ifadelerin içinde geçen veya daha önce işleme sokulup sonucu belirlenmiş olan “x==3”, “y<=4” gibi ifadeleridir. İşlemci çekirdekleri ne kadar hızlı olursa olsun, Bu ifadeler sıradan matematiksel veya pointer işlemlere göre bir nebze yavaş kalmaktadır. Özellikle string gibi dizi benzeri değişken türlerinde daha da yavaş karşılaştırılma sonuçlarına erişilmektedir. Ayrıca Her karşılaştırma işleminin cevabı sadece ve sadece bir bitlik veri olarak saklanmaya müsait durumdadır. Fakat işlemci teknolojilerinde bu veri işlemcinin kaç bitlik uzunlukta olduğuna göre sonuçlar üretmektedir. Tam olarak bu çalışma bu ifadeleri bir bitlik veriler halinde saklamayı ve sadece bir bitlik işlemler ile kullanmayı hedeflemektedir.

3. UYGULAMA VE TASARIM TERCİHLERİ

Bu bölümde çalışma boyunca kullanılacak uygulamalar, programlar ve seçilen mimari tercih sebepleriyle birlikte izah edilecektir. Bu izah sonra yapılacak olan tasarımlar ve yazılım kodları, belirtilen uygulamalar ve diller ile hazırlanacaktır.

3.1.MIPS Tercihi

İşlemcileri ele alırsak birçok mimari bulunmaktadır. Bunlar arasında CISC ve RISC olmak üzere iki temel mimari yapısı bulunmaktadır. CISC mimarisi işlemcinin işlemleri register array yerine, birkaç tanımlı özel register ve bellek arasında yapmayı tercih eder. RISC ise işlemlerini register'lar vasıtasıyla gerçekleştirip bellek iletişimini sadece veri taşıma olarak sınırlandırır. CISC mimarisi fazla sayıda komut ile çalışır. Komutlar genellikle bir register'ı hedef alır. Komut kümesinin genişliği entegrasyonu zorlaştırmaktadır. Zira komut kümesinin fazlalığı yeni komutların entegrasyon işlemlerini zorlaştırmaktadır. Entegrasyorunu zorlaşmasının sebebi şöyle açıklanabilir; Çok fazla komutun olduğu bir komut stine yeni bir komut eklemek için komut setinde makine kodları arasında yeni komutlara yer bulabilmek daha zor olacaktır. Ayrıca bu çalışmada bitler arası işlemin öneminden bahsedilirken bellek ile çalışma yapıldığında bitsel çalışma zor olacaktır. Çünkü bellekler bit olarak değil byte (8 bit) veya byte katları olarak tasarlanmaktadır. Bit olarak tasarlanması durumunda matematiksel işlemler zorlanacaktır.

MIPS işlemci mimarisi bir RISC mimarisidir. 32 farklı register ile çalışmakta, erişilmek istenen register adresi ile kullanılmaktadır. Ayrıca register üzerindeki değer bellek için potansiyel bir adres görevi görmektedir. Diğer bir açıdan MIPS mimarisini ele alırsak günümüzde kullanılan mimarilere kaynak durumunda bulunmaktadır. Örnek olarak ARM mimarileri temelinde MIPS mimarisine benzemektedir. Bunun sebebi MIPS mimarisinin open source bir mimari olmasıdır. Geliştirmelere açık olmasının yanı sıra MIPS mimarisi geliştirmelere açık bir mimaridir. Bu konuda çeşitli geliştirmeler de mevcuttur. MiniMIPS adlı bir çalışmada MIPS mimarisinden esinlenerek 8 bitlik bir eğitim amaçlı işlemci tasarlanmıştır. [1]

MIPS mimarisi 32 bit ve 64 bitlik farklı mimarilerde tasarlanmıştır. Bu çalışma esnasında 32 bit MIPS mimarisi örnek alınacaktır. Bunun sebebi ise bu çalışmanın net bir mimari modeli oluşturma değil bu çalışmadan sonraki çalışmalar için yol gösterici olma

hedefidir. Bu hedefi gösterirken karmaşık 64 bitlik bir mimari yerine kısmen daha basit 32 bitlik mimari tercihi uygun bulunmuştur. Yol gösterici olmada en büyük hedef çalışmayı MIPS ile sınırlı tutmak yerine tüm yeni tasarımlarda bu yol gösteriminin uygulanmasını misyon edinmektir. Günümüzde özellikle mikro denetleyici alanında 8 ve 16 bitlik mimariler bulunmasına rağmen bu mimarilere de entegrasyon 32 bit kadar kolay görünmemektedir. Çünkü 32 bitlik komut setinde, komut entegrasyonu için boş alan bulmak daha kolay olacaktır. 64 bitlik bir MIPS mimarisinde komutlar için daha fazla boşluk bulmak mümkündür, fakat uygulamanın imkanlarının kolaylığı, üzerinde yapılmış çalışmaların çokluğu ve çekirdeğin daha basit olması sebebiyle bu çalışma 32 bit olarak tasarlanmış MIPS mimarisinde uygulanması uygun görülmüştür.

3.2.FPGA Kullanımı

FPGA Donanım tanımlama dilleri (hardware description language - HDL) sayesinde elektronik devrelerin gerçekleştirilmesini sağlayan ortamdır. Donanım tanımlama dilleri ile yazılmış olan devre tasarımı burada uygulamaya müsaittir. Bu sayede testler ve uygulama hızlı olur ve maliyetlerin artmasını engeller. Bulunan hatalardan sonra yapılan düzeltmeler malzeme maliyeti oluşturmaz.

FPGA kullanımının diğer bir sebebi ise bu konuda yapılmış çalışmaların mevcut oluşudur. Çalışmamızda tercih olarak kullanılacak olan MIPS mimarisi ile ilgili çeşitli FPGA çalışmaları bulunmaktadır. Bir çalışmada Altera firmasının yazılımı olan QuartusII programında MIPS mimarisi 32 bit olarak ve detaylı bir şekilde anlatılmıştır. [2]

Farklı donanım tanımlama dilleri bulunmaktadır. Bu dillerden en fazla öne çıkanlar Verilog, VHDL dilleridir. Bu dillerin yanı sıra AHDL gibi dillerde mevcuttur. AHDL dili Altera firmasının geliştirdiği bir dil olup sadece bu firmanın cihazlarında derleyicisi mevcuttur bu yüzden her cihazda çalışmaya uygun değildir.

3.3. Verilog Dili ve Tercihi

Verilog, başka bir deyişle Verilog HDL dili daha önceki konuda bahsedildiği üzere bir donanım tanımlama dilidir. Verilog dili C programlama diline benzer bir şekilde

tasarlanmış olan bir dildir. C dilinden farklı olarak kıvrık parantezler yerine begin – end ifadelerini kullanarak çalışır.

Verilog dilinin çalışmamızda başlıca tercih sebeplerinden biri C diline benzemesi ve basit yazım formatıdır. Ayrıca verilog dilini hemen hemen her FPGA platformu desteklediğinden dolayı globalleşmiş bir dildir. Bu globallikten ötürü çok fazla kaynak bulunabilmektedir.

Yukarıda anlatılmış olan sebeplere istinaden bu çalışma kapsamında donanım tanımlama dili olarak Verilog HDL dili kullanılacaktır. Yapılacak tüm modül ve uygulamalarda bu dil esas alınacaktır.

3.4. Logisim ile Devre Tasarımı

Logisim uygulaması Java yazılım diliyle Dr. Carl Burch tarafından geliştirilmiş, sayısal devre tasarım ve test uygulamasıdır. [3] Bu uygulamanın boyutu 2.7MB civarında olup Java dilinin desteklediği her işletim sisteminde başarılı bir şekilde çalışmaktadır. Uygulamada devre tasarım anında canlı olarak sonuçlar gözükmektedir. Lojik kapılar ve kablo işlemlerinin yanı sıra bellekler plexer'lar , gişriş çıkışlar ve aritmetik işlemler gibi hazır devre elemanları da bulunmaktadır. Modüler çalışma mantığına sahip olan Logisim uygulaması, tasarladığınız devreleri diğer devreler içerisinde kullanmaya da olanak tanır. Ayrıca bellek ve çoklu giriş/çıkış birimi içermeyen devrelerde giriş çıkış analizi, kanough haritası ve devre sadeleştirme gibi özelliklerle kullanıcının işini kolaylaştırmanın yanı sıra devrenin hızlı sonuçlar vermesini de sağlamaktadır. Ayrıca bu sadeleştirme işlemleri sırasında kapıların giriş sayısının 2 ve tüm kapıların NAND kapısı olarak tasarlanmasında mümkün kılınmıştır.

Öğrencilerin FPGA ve Logisim ile öğrenmeleri üzerine yapılan bir çalışmada tüm alanlarda Logisim'in daha başarılı olduğu tespit edilmiştir. [4] Bu da Logisim'in kullanım ve anlam olarak başarısını ortaya koymaktadır.

Çalışma çerçevesinde Logisim uygulamasıyla devre tasarımları yapılacak ve bu tasarımlar üzerinden çeşitli testler oluşturulup içeriğe devre tasarımları ve sonuçları eklenecektir.

4. ENTEGRASYON

Bir işlemciye yeni birimler eklemek yani entegrasyon yapmak için o mikro işlemcinin mimari tasarımının entegrasyona uygun olması mecburidir. Ayrıca tasarlanacak olan birimlerinde mevcut işlemcinin mimarisine uygun tasarlanması şarttır. Bu sebeple işlemci mimarisi dikkatli incelenmeli ve birim tasarımları uygun bir şekilde ayarlanmalıdır. Birimler ayarlanırken komut setinin tasarımı da gerekmektedir. Çünkü işlemci mimarilerinde komutlar ve bu komutlara karşılık gelen makine dili sonuçları mimarinin gelişimi açısından önemli yer arz etmektedir.

MIPS mimarisi yeni birimlerin eklenmesine uygun bir pozisyondadır. Bunun başlıca kanıtlarından birisi MIPS mikroişlemci mimarisine float-point işlem biriminin entegre edilmesi olarak gösterilebilir. Entegrasyon işleminde genellikle hedef işlemciyi özelleştirmek, hızlandırmak veya komut setini geliştirmek olacaktır. Özelleştirmede amaç işlemci mimarisinden yeni bir işlemci modeli çıkararak bu sayede belirli konularda daha hızlı ve becerikli işlemciler üretmektir. Örnek olarak ekran kartlarını verebiliriz. Ekran kartları temelde birer işlemci mimarisi olup amaçları görüntü oluşumunu ana işlemciden bağımsız olarak sağlamak ve hızlı sonuçlara erişmektir. Günümüzde özellikle üç boyutlu oyun programlama alanında ekran kartları önemli yer arz etmektedir. Ayrıca çeşitli uygulamalarda yapılan animasyonlu görseller ekran kartları sayesinde CPU'nun yavaşlamasını engelleyerek hızlı sonuçlar elde etmeyi sağlamaktadır. Bir MIPS işlemciyle ekran kartı tasarımı güzel bir fikir olabilir fakat üzerinde çalışmakta olduğumuz çalışmanın hedefinde bu kesinlikle yoktur.

MIPS işlemciler üzerinde çeşitli entegrasyon yöntemleri denenmiştir. Bir örenke verilmek istenirse MIPS çekirdeği özelleştirilerek H264/AVC video codec'ine dönüştürülme işlemi hakkında veriler mevcuttur. [5] Başka bir örnekten bahsetmemiz gerekirse LDCP dekoderi oluşturma amacıyla yapılan bir çalışmada MIPS çekirdeği özelleştirilmiştir. [6] Bu tez çalışmasında örneklerde gösterilmiş çalışmalar gibi özelleştirmeler yapılmayacak fakat örneklere benzer şekilde entegrasyon işlemleriyle genel bir işlemci çekirdeği tasarlanacaktır. Bu çalışma şekli özelleştirme değil MIPS işlemci çekirdeğini geliştirme amacıyla yapılmaktadır. Bu yapıyı float-point işlem birimine benzetmek mümkündür.

4.1. Instruction Fetch (IF) ve Instruction Decode (ID) Üzerinde Geliştirmeler

IF ve ID birimi MIPS mimarisi için komutları yükleyen ve sistemde doğru ünitelerin çalışmasını sağlayan birimlerdir. Bu birimler sayesinde komutlar okunur ve gerekli birimler aktif edilerek işlemler gerçekleştirilir. IF aynı zamanda pipeline işlemlerine bağlı olarak bir sonraki komutun okunup okunmaması konusunda kararlar alır. Bu sayede pipeline desteği olmayan veya adres çakışmasına sebep verilecek işlemler doğru çalışma zamanı gelinceye dek IF üzerinde bekletilir. Doğru çalışma zamanında kasıt komutun bir önceki komut ile bağının olup olmaması yani aynı bir önceki komutun değiştireceği bir register'ın kullanılıp kullanılmadığı veya bir önceki komutun program akışında sonraki komutların çalışıp çalışmaması konusunda ektisi olup olmadığıdır. Eğer böyle bir etki oluşacaksa programın önceki komutu beklemesi doğruluk açısından vazgeçilmez bir unsurdur.

MIPS mimarisinde geliştirme yapma konusunda IF ve ID birimleri önemli rol oynamaktadır. Her ne kadar tüm değişiklikler bu birimlere bağlı yapılma gereksiniminde bulunmasa da (örnek olarak ALU üzerinde bir işlemin hızlandırılması IF veya ID üzerinde değişiklik gerektirmez) özellikle komut setini etkileyecek değişiklikler bu birimlerin değişikliğiyle mümkün olmaktadır. MIPS mimarisinde IF ve ID birimleri üzerinden yapılan değişiklilere ilk örnek olarak çalışmamız gösterilemez. Zira bu konuda farklı alanlarda yapılmış çeşitli çalışmalar bulunmaktadır. Örnek vermek gerekirse güç tasarrufu hedefli yapılmış olan bir MIPS üzerinde geliştirme makalesinde IF ve ID birimlerinde çeşitli değişiklikler yapılmıştır. [7] Bu çalışmalar MIPS üzerinde bulunan IF ve ID birimlerinin geliştirmeye açık olduğunun bir kanıtıdır.

IF biriminin amacı komutları bellekten okuyarak bu komutlara uygun davranışı göstermektir. Daha önce bahsettiğimiz üzere pipeline işlemleri IF üzerinde bekletmeler sayesinde doğru bir şekilde sonuçlar vermektedir. MIPS açılımlı haliyle Microprocessor without Interlocked Pipeline Stages, pipeline sayesinde kendisinden önce tasarlanmış olan mimarilerle farkını ortaya koymuştur. Bu sebeple yeni eklenecek komutlarda doğru bir pipeline desteği önemli yer tutmakta ve bu sebeple IF biriminin geliştirilmesine ihtiyaç duyulmaktadır.

ID biriminin amacı komutları çözerek birimleri aktif etmektir. ID birimi üzernde yapılacak geliştirmeler ile çalışma üzerinde anlatılacak olan dört farklı birim, MIPS

çekirdeğine entegre edilebilir olacaktır. ID birimi sayesinde komut setine yeni eklenecek olan komutlar, yeni eklenecek olan birimleri gerektiği durumlarda aktif edecektir.

4.2. Pipeline

Pipeline bir mikroişlemci çekirdeğinin komutları aşamalar ile çalıştırıp önce gelen komutun bir sonraki aşamaya geçtiğinde önceki aşamaya bir sonraki komutun alınıp çalıştırılmasıdır. Pipeline mikroişlemci performansında etkili sonuçlar gösteren bir mantıksal uygulamadır. Örnek olarak MIPS mimarisinde Pipeline'ı inceleyelim. MIPS mimarisi 5 farklı aşama ile komut çalıştırmaktadır. Bunlar IF, ID, EX, MEM, WB olarak adlandırılır. [7] Birinci komut önce IF noktasına gelir. Bu noktayı geçip ID noktasına geldiğinde ikinci komut IF noktasına alınır. Bu şekilde beş aşama tamamlanır. Yani çalışma anında aynı anda beş farklı komut çalıştırılabilmektedir. [8] Bu sayede performans arttırılır. Eğer MIPS bu yapıda olmasaydı yani aynı anda sadece bir komut çalıştırılsaydı, performans yaklaşık olarak beşte birine düşecekti. Kesin olarak beşte birine düşmemesinin sebebi ise art arda gelen her komutun pipeline ile çalıştırılamamasıdır. Bu şekilde çalıştırma yapıldığında sonuçların yanlış olma ihtimali yüksektir. Örnek olarak aşağıda bulunan komut satırlarını ele alalım:

```
add $s1, $s2, $s3

add $s4, $s5, $s6
```

Bu komutlardan birincisinde sonuç $s1 adresli register'da saklanacaktır. İkinci sırada bulunan komuttaysa $s1 kullanılmamaktadır bu yüzden pipeline art arda çalıştırılabilir.

```
add $s1, $s2, $s3

add $s4, $s1, $s6
```

Bu komutlardan ikincisinde görüldüğü gibi birinci komutun sonucu yani $s1 kullanılmıştır. Bu yüzden ikinci komut birinci komutun sonucunun yazılmasını beklemek durumundadır. İkinci komut IF aşamasına girdikten sonra birinci komutun WB aşamasına gelmesini bekler.

Ayrıca her komut Pipeline desteğine sahip değildir. Bazı komutlar girdiğinde yeni komut alımının komut sonuçlandırılıncaya kadar beklemesi şarttır. Örnek olarak dallanma işlemlerini varsayabiliriz. Eğer direkt veya karşılaştırmalı bir dallanma komutu gelmişse farklı bir adrese gidileceğinden dolayı bir sonraki satırın çalışmama ihtimali vardır.

İleride anlatılacak olan BLU biriminin sisteme eklenmesinde pipeline desteği verilmesi gerekmektedir. Bu destek verildiğinde pipeline'nın beklemesine sebep olan register adresleri azalacaktır. Bu pipeline işleminin aktif olmasını sağlayacaktır. Bu register'ların azalmasının sebebi karşılaştırma sonuçlarının BRA birimine yazılacak olmasıdır. Bu birime yazılmadan dolayı registerlar daha aktif kullanılacak ve bu sayede komutlar arasında ortak register sayısı azalacaktır. Bunun neticesi pipeline yapısında artışı ve doğal olarak işlemcide komutların daha önce çalıştırılması sağlanacaktır.

Yeni eklenecek komutlardan BLU ile çalışan komutlar ve RA ile BRA arasında veri taşıma komutları yukarıdaki mantıkta adres çakışmadığı müddetçe pipeline işlemlerini desteklemeye müsaittir. Bu konuda her hangi bir sorun oluşmayacaktır. Dallanma işlemi yapan komutlar ise işlem bitene kadar pipeline yapısını beklemeye alacaktır. Ayrıca eklenecek karşılaştırma komutları da üstte anlatılan mantıkta pipeline desteğine sahip olabilecektir.

5. BİTLERİN SAKLANMASI

Bitler ile yapılan işlemlerin öneminde dolayı bitler ile çalışmak ve bu bitleri saklamak gerekmektedir. Önceki bölümde üzerinden geçilen BRA birimi bitleri saklamayı sağlayan ve mimariye entegre edilecek bu çalışmadaki en önemli birimlerin arasında gelmektedir.

5.1. Bit Register Array (BRA)

Bit register array'in görevi çeşitli işlemler sonucu elde edilen bit değerlerini saklamaktır. Ayrıca bu bitler sayesinde dallanma işlemleri gerçekleştirilir. BRA MIPS çekirdeğinde bulunan Register Array ile aynı yapıda olup 32 adet bir bitlik veri saklayacaktır. Bu veriler okuma anında iki farklı çıkış ile okunabilir. Yazma anında ise sadece bir bit yazılabilir. Ayrıca tasarlanacak işlemci ve derleyicinin yapısına göre Zero, Negative, Overflow, Interrupt gibi bayraklar da BRA içerisinde saklanabilir.

BRA üzerinde 2 okuma (giriş) 1 yazma (giriş) değerinin olma sebebi MIPS işlemcilerin register işlemlerinde üçlü adres modu kullanmasıdır. Yani İki register'dan alınan bilgiler işlenir ve sonuç bir register'a yazılır. Aynı şekilde BRA üzerinde yapılacak işlemlerde de iki bit alınıp işlem sonucu bir bite kaydedilecektir.

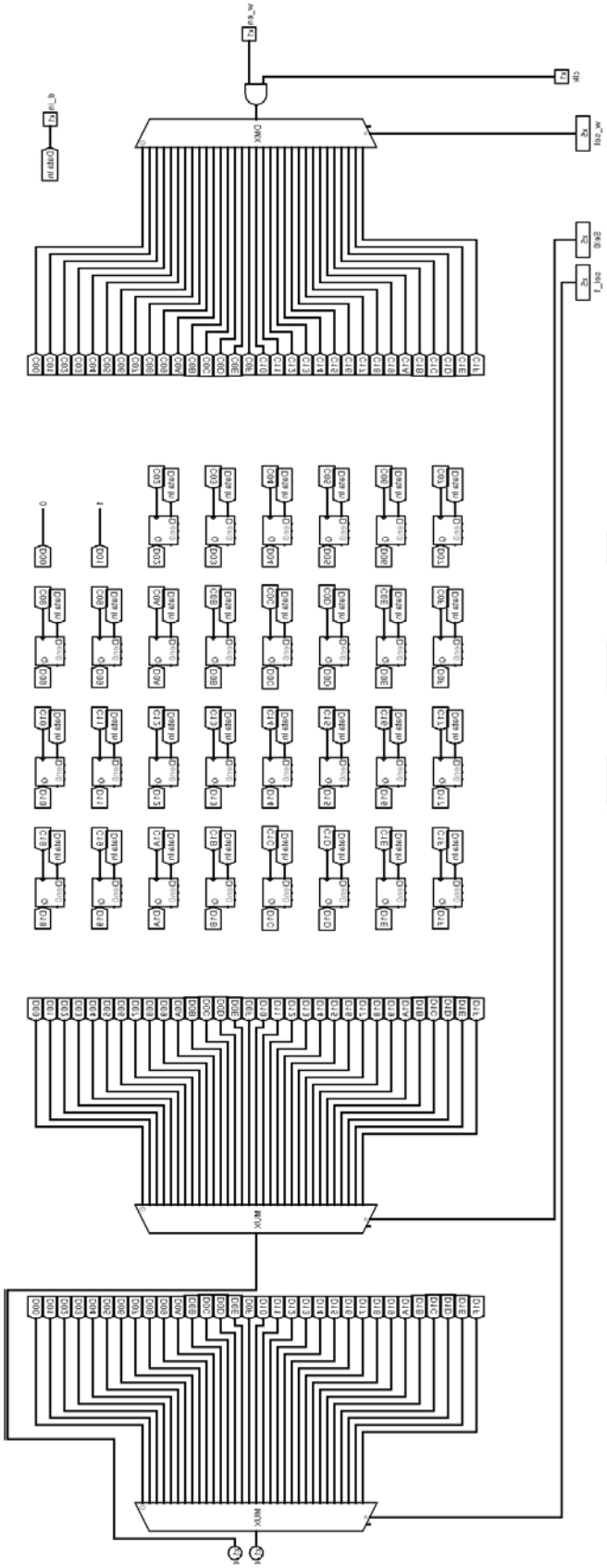
BRA biriminin 0 adresli bitinin değeri sıfır olmalıdır ve hiç bir şekilde değiştirilmemelidir. Bunun sebebi MIPS register'larında olduğu gibi sıfır değeriyle işlem yapılabilmesidir. MIPS register'larında sabit 1 değeri taşıyan hafıza yoktur. Çünkü Register'lar 32 bit olduğundan dolayı 0 değeri dâhil 4.294.967.296 farklı değer alabilir. Bu değerlerden 0 karşılaştırma işlemlerinde kullanılabilir. Bu yüzden aktif olarak 0 değeri sabit tutulmuştur. BRA Register hafızalarında bir bitlik veri tutulduğundan dolayı 0 ve 1 olmak üzere sadece iki değer alabilir. Bu yüzden bu iki değer birer register'da sabit olarak saklanabilir. 0 biti 0 adresli regiter'da 1 biti ise 1 adresli register'da tutulur. Bu sabit 0 ve 1 registerlarının işlemciye faydası ise bir bititn 0 veya 1 ile işleme girmesi gerektiğinde kolayca kullanılabilmeleridir. Bunun yanısıra eğer bir bite 0 veya 1 değeri atanmak istenirse bu bitler ile aktarma işlemi yapılabilir ve bu bitler ile mutlak dallanma işlemleri de yapılabilir olacaktır. Mutlak dallanma işlemleriyle alakalı bilgiler dallanma bölümünde detaylı bir şekilde aktarılacaktır.

MIPS işlemcilerde işleme sokulacak tüm değerler (veri taşıma işlemleri hariç) register'lar üzerinde tutulmaktadır. Bir karşılaştırma işlemi yapıldığında sonuç değeri ancak ve ancak bir register ile saklanır. BRA birimi eklendiğinde karşılaştırma işlemlerinin sonuçları burada saklanacak ve register'lar bu kullanımdan uzak tutulmuş olacaktır. Bu sayede daha fazla register varmışçasına işlemcinin performansına etki edecektir. Etki Register ihtiyacı olması durumunda belleğe kopyalamaların azalması ile kendini gösterecektir.

BRA üzerinde bir bitlk verilerin saklanmasından dolayı burada register'lar bir bit olacaktır. Yani sadece bir flipflop olması yeterlidir.

5.2. Bit Register Array Biriminin Tasarımı

Şekil 1'de BRA biriminin logisim ile yapılmış olan tasarımı bulunmaktadır. Tasarım D flipflop'lar ile yapılmıştır. Kablo karmaşasını engellemek amacıyla Logisim uygulamasının araçlarından olan tunnel kullanılmıştır. MIPS mimarisine uygun tasarım yapılması ve ihtiyacın bu mimaride net olmamasından dolayı bayrak değerleri tasarımda BRA birimine eklenmemiştir. Ayrıca tasarım görseli normal şartlarda yatay olup okunurluğun artması amacıyla çalışmanın içinde dikey olarak gösterilmiştir.



Şekil 1. Bit Register Array Biriminin Logisim Tasarımı

Şekil 1'de BRA biriminin Logisim uygulaması kullanılarak yapılmış olan çizim gösterilmektedir. Bu çizimde bulunan girişler şöyledir.

- 1 bit kayıt verisi girişi (d_in)
- 1 bit yazma aktif edici giriş (w_en)
- 1 bit clock (clk)
- 5 bit yazma selektörü (w_sel)
- 5 bit 1. okuma selektörü (sel_0)
- 5 bit 2. Okuma selektörü (sel_1)
- 1 bit 1.veri çıkışı (d_0)
- 1 bit 2.veri çıkışı (d_1)

sel_0 ve **sel_1** verileri dışarı çıkarılacak olan bit değerlerinin adresini belirler. **w_sel** ise yazılacak olan bitin adresini belirler. Yazılacak değer ise **d_in** ile içeri alınır. **w_en** biti yazmanın aktifliğini ayarlar. Eğer bu bit bir olursa yazma işlemi sağlanır. **clk** biti frekans ile gelen saat darbesidir. **d_0** çıkışı **sel_0** ile adres verilen bitin sonucunu, **d_1** çıkışı ise **sel_1** ile verilen adresin değerini dışarı çıkarmayı sağlar.

Aşağıdaki kod bloğunda ise BRA biriminin verilog dilinde yapılmış olan tasarımı bulunmaktadır.

```
module BRA(d_in,w_en,w_sel,sel0,sel1,d0,d1,clk);
input d_in,w_en,clk;
input[4:0] w_sel,sel0,sel1;
output d0,d1;
reg d0,d1;
reg[31:0] regs;

    always begin
    case (sel0)
        00:d0<=1'b0;
        01:d0<=1'b1;
        default: d0<=regs[sel0];
    endcase
    end
    always begin
    case (sel1)
        00:d1<=1'b0;
        01:d1<=1'b1;
        default: d1<=regs[sel1];
    endcase
    end

    always @(posedge clk) begin
        if (w_en && w_sel != 5'd0&&w_sel!=5'd1)
egin
            regs[w_sel] <= d_in;
        end
    end
endmodule
```

6. BİTLER ARASI İŞLEMLER

İlk yapılan mikroişlemcileri düşünürsek birebir bitler arası işlemler ihtiyaç olarak görülmemekteydi. Bunun sebebi ise nesneye dayalı programlama anlayışının olmaması ve derleyicilerin C dili ve bu seviyedeki ilk yazılım dilleri için hazırlanmasından kaynaklanmasıydı. C dili boolean veriler üzerinde çalışmadığından dolayı bitler arasında işlem yapılması yani boolean işlem yapılması mantıklı değildir. Fakat C++ ve daha sonra geliştirilmiş olan birçok yazılım dilinde boolean kavramı önemlidir. Bu dillerde if else gibi komutlar boolean bir sonuca göre çalışmaktadır. C dilinde ise boolean kavramı yerine tamsayı olan bir değerin sıfır olup olmamasına bakılır. Yani C dilinde karşılaştırma işlemleri sonucu sıfır veya bir sonuçlarıyla tam sayı olarak saklanır. Bu iki değerden farklı olan bir tam sayısı if komutunda kullanılırsa bunu bir değeri gibi yani doğru olarak kabul edecektir.

Boolean işlemlerin eğitim açısından da önemi yüksektir. Çünkü işlemci mimarilerinin temelini boolean cebri oluşturmaktadır. Bu konuda eğitimin nasıl olması gerektiğini anltan bir çalışmalar bulunmaktadır. Yapılmış olan bir akademik çalışma yüksek lisans seviyesindeki öğrenciler için boolean cebri, FPGA ve MIPS mimarisi gibi konularda eğitim yapısı ve laboratuvar çalışmaları hakkında yeni fikirler sunulmaktadır. [9] Boolean cebri ve işlemci mimarisi bağlantısı sebebiyle işlemci üzerinde boolean işlemler yapan bir birimin veya birimlerin olması, bu konuda eğitim alan öğrencilerin gelişimi açısından faydalı olacaktır.

Boolean yapısıyla alakalı olarak Boolean Programs çalışması bulunmaktadır. Bu çalışmada tüm değişken türlerini boolean olarak kabul eden bir yazılım dili tasarımı yapılmıştır. [10] Bu yazılım diline diğer dillerden kod çevirme işlemleri yapılabilmektedir. Örnek olarak SATABS uygulaması gösterilebilir. Bu uygulama C dilinden Boolean Programs yapısına uygun çeviri yapmaktadır. [11] Farklı bir uygulama olarak Microsoft firmasının geliştirmiş olduğu SLAM adlı uygulama bulunmaktadır. [12] Bu uygulama yazılımın çalıştığı arabirimleri kontrol etmek için kullanılmaktadır. Boolean Programs ile alakalı bir çalışmada ise Boolean Programs yapısının otomatikleştirilmesiyle alakalı model oluşturulmuştur. [13]

Boolean veriler işlemciler üzerinde C dilinin mantığına uygun olarak tamsayı olarak saklanır. BRA bölümünde bahsedildiği üzere bu verilerin bir bitlik hafızada tutulması mantıklı olacaktır. Ayrıca bu bitler arasında programlama dillerinde işlem yapılabildiğini

düşünürsek bu işlemlerin aynı şekilde yapılması ihtiyaç olacaktır. Bu ihtiyaçtan dolayı ise BLU birimine ihtiyaç vardır.

6.1. Bit Logical Unit (BLU)

BLU biriminin amacı daha önce anlatılar Bit Register Array (BRA) biriminde bulunan bitler arasında lojik işlemler yapmaktır. Mantık olarak ALU ile aynı mantıkta çalışır. MIPS ALU yapısında olduğu gibi iki işlem girişi ve bir sonuç çıkışı vardır. Bu giriş ve çıkışlar 1 bit olarak tanımlanır. BLU da yapılan işlemden sonar oluşan sonuç BRA'da belirtilen adrese kaydedilir. Bu adres daha önce belirttiğimiz gibi 0 veya 1 olamaz. Çünkü 0 veya 1 adresleri sabit olarak tanımlanmıştır, 0 ve 1 bilgilerini sabit olarak saklamaktadırlar. Ayrıca işlemci çekirdeğinin bayrakları bu BRA'da önerildiği gibi saklanırsa bu saklama bitlerine sonuç verisi kaydedilemez.

MIPS işlemcilerde ALU'da 4 bitlik işlem seçimi bulunur. BLU birimi de bu sebepten dolayı 4 bitlik bir sayıya sahip olabilecek imkândadır. Çünkü BLU üzerinde yapılacak olan işlemler ALU üzerinde yapılacak işlemlere benzer şekilde komutlarla çalıştırılabilir olmalıdır. 4 bitlik işlem birimi tasarımda 2^4=16 farklı işlemi destekler. Fakat bitler arasında 16 farklı işlem tanımlamak zorunlu değildir. Daha az sayıda işlem ile çalışma yapılması mümkündür. Bitler arasında genel yapılabilecek lojik işlemler şöyledir:

1. AND
2. OR
3. NOT
4. XOR
5. XNOR
6. NAND
7. NOR

Bu yedi işlemin tamamı BLU birimine konulabilir fakat programlama için hepsi gerekli değildir. C veya benzeri bir yazılım dilinin derleyicileri bu işlemlerin tamamına ihtiyaç duymayabilir. Eğer işlem sayısı 4 ile sınırlandırılırsa selektör iki bit sonucu seçimi yapan multiplexer küçültülüp transistor miktarında azalma ve çalışma hızında kazanç sağlanabilir. Fakat bu durumun gerekliliğini sorgularsak şu sonuçlara ulaşırız.

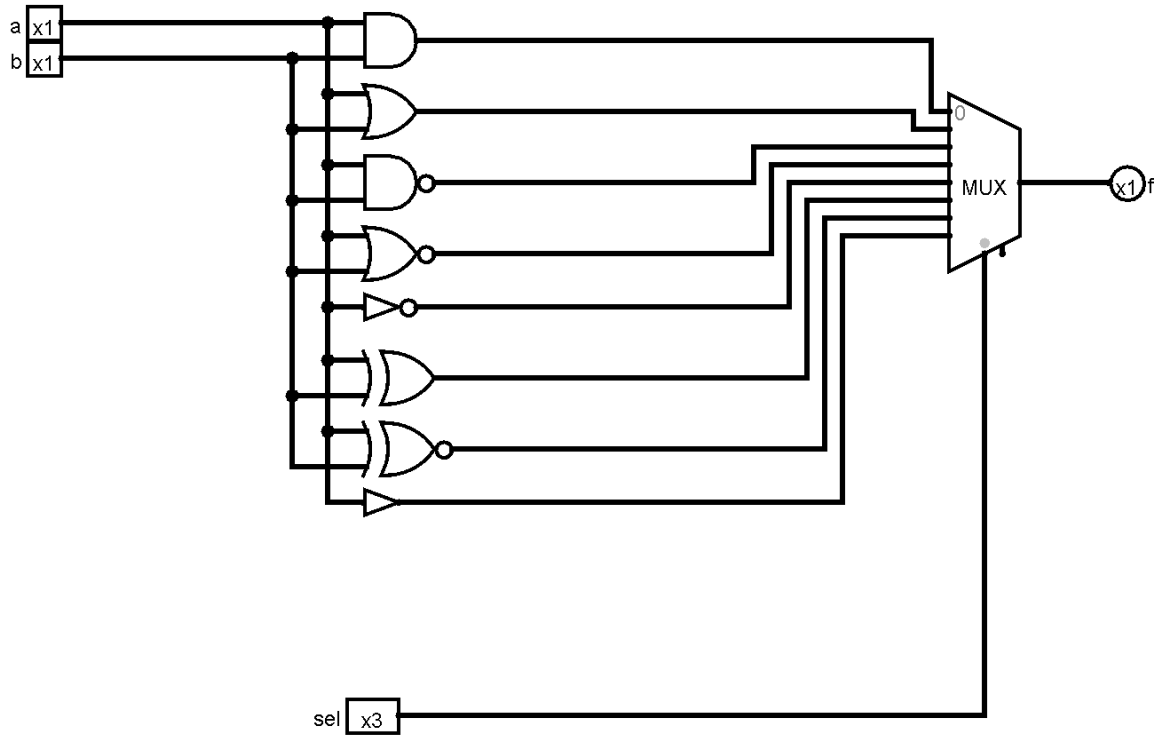
1. İşlemciler milyarlarca transistörle tasarlanmaktayken birkaç transistor kazanmak kârlı olabilir mi?
2. Çarpma Bölme gibi ağır işlemleri yapan bir MIPS işlemcisinde bir bitlik işlemlerin frekans olarak bir kârı olur mu?

Birinci sorunun cevabı şöyledir: MIPS işlemcisi yeni nesil işlemcilerde olduğu gibi çok fazla transistor kullanmaktadır ve 2 bite göre tasarlanacak multiplexer bize görünü bir kâr elde ettirmeyecektir. İkinci sorununun cevabına gelirsek işlemcinin frekansı yapılan en uzun işleme yani süre olarak en geç cevap veren işleme göre yapılacaktır. Bu cevap süresi o işlemde kullanılan seri transistor sayısına bağlı olarak tespit edilmelidir. Yukarıda görülmekte olan yedi işlemin tamamında ise maksimum 4 seri transistor bulunabilir. Bu da bu işlemlerin azaltılmasının hızı etkilemeyeceğini net olarak göstermektedir.

İşlemlerin tamamının görünür bir faydası olmamasına rağmen (örnek olarak MIPS işlemciler NAND işlemini barındırmaz ve bu yüzden MIPS derleyicilerinde NAND işleminin karşılığı yoktur.) daha sonra oluşturulacak bir derleyicide aktif olarak kullanılabilir. Fakat bu konu bu makalenin konusu değildir. Bu yüzden bu makalede anlatılan işlemcide belirlenen yedi işlemin tamamı kullanılacaktır. Yedi işlem kullanılması durumunda ise $2^3 = 8$ olacağından dolayı multiplexer üzerinde bir giriş boş olacaktır. Bu biti değerlendirmek şart değildir fakat şöyle bir kullanım şansı olabilir. Eğer bu bit girişi ilk gelen değeri direkt veya bir buffer aracılığı ile aktarırsa bu giriş değerlendirilmiş olur ve bit transferi görevini üstlenir. Yani herhangi bir bit başka bir adrese kopyalanmak istenirse BLU üzerinden sağlanmış olacaktır. Bit transferi için AND OR işlemleri kullanılabilir. Örnek olarak 0 adresli bit değeri daima 0 olduğu için kopyalanmak istenen bit bu bitle OR işlemine tabi tutulabilir. Veya direkt olarak kendisiyle OR işlemine tabi tutulabilir. Fakat çalışma bazında program koduna dayalı okunurluğun artması planlandığından dolayı BLU üzerinden kopyalanması işlem olarak hız kazandırmasa dahi kod okunurluğunu etkileyecektir. Çünkü kullanıcıya OR işlemi göstermek yerine direkt transfer işlemi göstermek, işlemin veri transferi olduğunu göstermek açısından daha efektif olacaktır.

6.2. Bit Logical Unit Tasarımı

Logisim uygulamasıyla yapılmış olan tasarım aşağıdaki Şekil 2'de gösterilmiştir. Bu tasarımla ilgili detaylar görselden sonra anlatılacaktır.



Şekil 2. Bit Logical Unit biriminin Logisim Tasarımı

Şekil 2'de Logisim ile tasarlanmış olan BLU birimi gösterilmiştir. Giriş ve çıkışlar aşağıdaki listede bulunmaktadır.

- 1 bit işlem yapılacak birinci veri (a)
- 1 bit işlem yapılacak ikinci veri (b)
- 3 bit işlem selektörü (sel)
- 1 bit sonuç (f)

a ve **b** bitleri arasında sekiz işlem yapılıp, 3 bit selektör girişi ve 8 bit veri girişi bulunan 1 bitlik bir multiplexer yardımı ile seçilen işlem sonucu **f** çıkışı ile dışarıya aktarılmaktadır.

Bu birimin verilog donanım tanımlama dili ile tasarımı ise aşağıdaki kod bloğunda gösterilmiştir.

```
module Blu(A,B,Sel,F);
input A;
input B;
input[2:0] Sel;
output reg F;

always
 begin
  case(Sel)
  3'b000:F<=A&B;     //AND
  3'b001:F<=A|B;     //OR
  3'b010:F<=~(A&B); //NAND
  3'b011:F<=~(A|B); //NOR
  3'b100:F<=~A;      //NOT
  3'b101:F<=A^B;     //XOR
  3'b110:F<=A~^B;    //XNOR
  3'b111:F<=A;       //Transfer
  endcase
  end
endmodule
```

7. VERİ TAŞIMA

BRA biriminin tasarlanmasından sonra en büyük ihtiyaç BRA birimine ve BRA biriminden veri taşımak olacaktır. BRA birimindeki veri direkt olarak belleğe aktarılabilir fakat bu büyük veriler tutan bellek birimi için çok mantıklı bir seçim olmayacaktır. BRA biriminde bulunan bir veri registerlara aktarılıp kullanılabilir. ALU birimi işlemleri gibi üç adres tipine uygundur. Birinci adres taşınacak bit adresi veya verinin saklanılacağı bit adresi, ikinci adres register adresi üçüncü ise register üzerindeki bitin indeksi olabilir. Registerda 32 bit olduğu için 5 bitlik bir adres ile kullanılabilir. Bu adres veri taşıması sırasında bir multiplexer yardımıyla bitin seçilmesini sağlayacaktır. Multiplexer Register'dan bite geçişte register'ın ilgili bitini seçip verilen adrese atacaktır. Bitten register'a geçişte ise ilgili register bitinin değerini belirleyerek diğer değerleri register'dan alacak ve sadece ilgili biti BRA'da verilen adresten çekecektir. Sonucu ise aynı bitin üzerine yazacaktır.

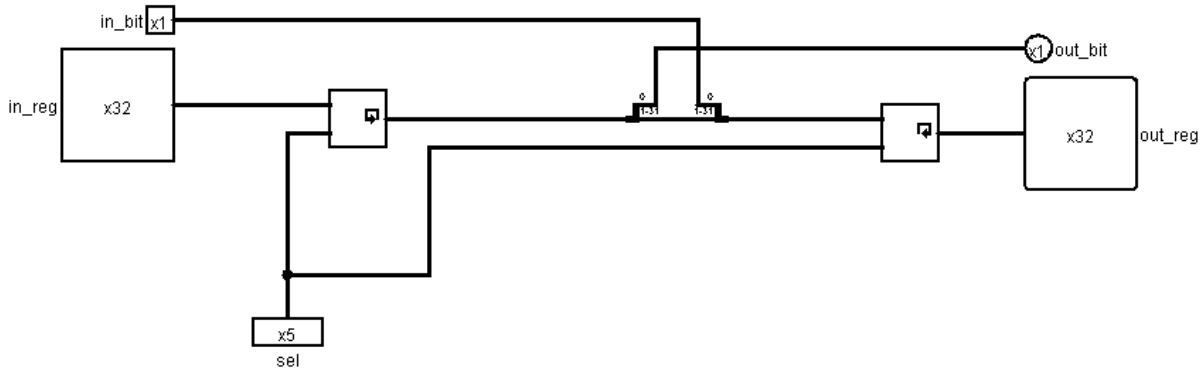
Başka bir yöntem ise register array üzerindeki bir register değerini tamamen BRA olarak tanımlamaktır. Yani BRA birimi sanal olacak ve registe array üzerinde bulunan 32 bitlik bir register vasıtası ile saklanacaktır. Böyle bir yapı oluşturulmaya müsait olsa da kesinlikle çok karmaşık olacaktır. Fakat Multithread olarak bir yazılım mimarisinde veya multi processing olarak çalışan sistemler için register'lar yedeklenmek istediğinde avantaj sağlayacaktır.

7.1. Bit Transfer Unit (BTU)

Bit seçim biriminin amacı register üzerinde bulunan herhangi bir bitin BRA biriminde saklanmasını sağlamaktır. Yani RA üzerinde bulunan ve adresi ve sırası komut içerisinde verilmiş olan bir bit değeri buradan kopyalanarak BRA üzerine kaydedilecektir. Bu sayede istenilen her hangi bir bit BLU biriminde kullanılabilecek ve/veya dallanma işlemlerinde görev alabilecektir. Bu birim bir bit selector yardımıyla register üzerinden gelen 32 bitlik veriden verilen sıra değerine göre biti çekecektir. Bu birim oldukça basit bir işlev yerine getirecek ve komutunda fonksiyon değerine ihtiyaç duymayacaktır. Bu birimi kullanan komutlar ise MIPS üzerinde R-Type komut olarak tanımlanacaktır. Çünkü R-Type komutlarda 5 bitlik 3 farklı adres verisi tutulmaktadır. Birinci adres bilgisi BRA üzerinde kaydedilecek biti, ikinci adres bilgisi RA üzerinde seçilecek register'ı üçüncü adres bilgisi ise register üzerinde hangi sıradaki bitin seçileceğini belirtecektir.

7.2. Bit Transfer Unit Tasarımı

Logisim uygulaması kullanılarak yapılmış ola BTU biriminin tasarımı Şekil 3'de gösterilmiştir.



Şekil 3. Register Transfer Unit Biriminin Logisim Tasarımı

Yukarıdaki logisim devresinde:

- 1 bit BRA değeri girişi (in_bit)
- 32 bit Register değeri girişi (in_reg)
- 5 bit selektör (sel)
- 1 bit BRA için bit çıkışı (out_bit)
- 32 bit Register değeri için çıkış (out_reg)

bulunmaktadır. **in_reg** değeri rotate right işlemi ile **sel** biti kadar döndürülmekte ve bu sayede 0'ıncı bit **sel** ile sırası belirtilen bit olmaktadır. 0' bit bu sayede **out_bit** olarak dışarı aktarılmaktadır. Daha sonra çıkarılan 0'ıncı bit yerine BRA üzerinden adresle gelen **in_bit** değeri 0'ıncı bit olarak eklenip **sel** değeri kadar rotate left işlemine tabi tutulmakta ve register değeri eski noktasına geri döndürülerek bit ekleme işlemi tamamlanmıştır. Komuta bağlı olarak **out_bit** değeri BRA üzerine kaydedilecek ya da **out_reg** değeri RA üzerine kaydedilecektir.

Aşağıdaki verilog kodunda BTU isimli birimin tasarımı mevcuttur.

```
module btu(in_bit,in_reg,out_bit,out_reg,sel);
    input in_bit;
    input[31:0] in_reg;
    input[5:0] sel;
    output out_bit;
    output[31:0] out_reg;
    reg out_bit;
    reg[31:0] out_reg;
    reg[31:0] transfer;
    always begin
        out_bit <= in_reg[sel];
        transfer <= in_reg;
        transfer[sel] <= in_bit;
        out_reg <= transfer;
    end
endmodule
```

8. KARŞILAŞTIRMA İŞLEMLERİ

Bilgisayar mimarisinde karşılaştırma işlemlerinin temelinde çıkarma işlemi ve bayraklar bulunmaktadır. Örnek olarak 32 bit büyüklüğünde işaretsiz iki A ve B sayısı birbirleriyle karşılaştırıldığında A sayısından B sayısı çıkarılır ve sonucunda oluşan bayraklar kontrol edilir. Eğer carry bayrağı 1 ise A sayısı B sayısından küçüktür. Eğer Zero bayrağı 1 ise iki sayı birbirine eşittir ve son olarak eğer bu iki bayrakta 1 değilse A sayısı B sayısından büyüktür denir. Bir işlemci çekirdeği direkt olarak bu bayrakları kullanmasa bile bu mantıkta işlemler ile sonuca varır. MIPS ise karşılaştırma komutlarıyla dallanma işlemi yapmaktadır. Örnek olarak BLEZ (Branch on less than or equal to zero) komutu belirtilen bir register üzerindeki değerin 0'dan küçük olup olmama durumuna bakar ve eğer sayı değeri 0'dan küçük bir değerdeyse dallanma işlemi yapar. [14] Fakat burada dikkat edilmesi gerek nokta bu dallanma işlemini 16 bitlik işaretli bir verinin PC değerine eklenmesi ile olur. Yani İşaretli olduğunu varsaydığımızda en fazla 2^{15}-1 ileri veya 2^{15} geri gidebilir. Bu da 127 satır ileri veya 128 satır geri gidebileceğini gösterir. Küçük döngü veya koşullarda bu derleyicide iş görmüş gözükmesine rağmen büyük işlemlerde tam istenilen sonucu vermeyecektir. Ayrıca bu dallanma komutlarından BNE ve BEQ haricindeki komutlar bir register ile 0 değeri arasında bulunan ilişkiye göre çalışmaktadır. [14] Yani bu dallanma işlemi yapılmadan önce kesinlikle karşılaştırma işlemi yapılıp sonucunun bir register'da tutulması ve bu sonucun sıfır karşısındaki durumuna göre işlem yapılması gerekmektedir. Bu da en az iki satırlık bir kod yazmayı gerektirmektedir. Örnek vermek gerekirse A sayısının B sayısından büyük olma durumunu kontrol edip büyük ise dallanma yapan bir kod yazmak istersek, birinci yöntem: A sayısından B sayısını çıkarıp sonuç register değerinin 0'dan büyük olduğunu kontrol etmektir. İkinci yönetim ise MIPS karşılaştırma komutlarıyla A'nın B den büyük olduğunu kontrol ettirip (eğer büyükse sonuç register değeri 1 olacak) sonuç register'ının 0 sıfıra eşit olup olmadığını kontrol etmektir. Bu iki durumda da en az iki satır kod yazılır ve her durumda bir adet sonuç register'ı gerekir. Oysaki ikinci yöntemde sonuç değeri 0 veya 1 olacaktır. Aslında bir bite sığan bir veridir ve bu veriyi MIPS 32 bitlik bir register ile saklamaktadır. Ayrıca genel amaçlı kullanılabilecek bir register sadece bir bitlik veri için kullanılıp register yetersizliği durumunda bellek kullanımına dahi sebep olabilecektir.

MIPS işlemcilerde 32 adet register bulunmakta ve bazı register'lar özel amaçla kullanılmaktadır. Özel amaçla kullanılan regiter'lara kayıt yapılamamaktadır. Bu yüzden kullanılabilir register sayısı daha da azalacaktır. Register sayısını 32'den 64'e çıkarmak

mükündür fakat bu durumda da kontrolü zorlaşacak, hızı yavaşlayacak ve komut yapısı değiştirilmek zorunda kalınacaktır. Komut yapısının değişmesiyle birlikte derleyici yapısı tamamen değiştirilmek zorunda kalacaktır. Tek bitlik veriler için tasarlanacak olan BRA birimi ise komut setine ekstra komutlar getirecek ve komut yapısını bozmayacaktır. Derleyici yapısında değişikliklere sebep olacak fakat bu değişiklikler derlenen programı basitleştirecektir.

Karşılaştırma işlemlerinin sonucu daima tek bitlik bir sonuç oluşturur. Bu makale bu sonucun BRA birimine kaydedilmesini planlamaktadır. BRA birimine kaydedilme işleminden sonra bitler arasında mantıksal işlemler yapılabilir olacaktır. Bu işlemlerin sonucunda dallanma işlemleri sadece BRA üzerinde bulunan bir bitin 1 veya 0 olmasına göre planlanacaktır. Yani dallanma işlemleri daha basit kod satırlarıyla kullanılmaya başlanacaktır. Ayrıca yukarıda bahsetmiş olduğumuz gibi karşılaştırma işlemlerine bağlı dallanma işlemleri genelde en az iki satır kod ile yapılabilmektedir. Önce bir karşılaştırma işlemi yapılmakta daha sonra ise bu işlem sonucu oluşan değere bakılmaktadır. Makale üzerinde karşılaştırma konusunda anlatılmak istenen ise bu işlemin sonucunun bir bitlik BRA biriminde saklanılması, işlenilmesi (ihtiyaç durumunda farklı işlemlere tabi tutulması) ve karşılaştırma için kullnılmasıdır. Ayrıca bayrak değerleri de bu birimde saklanırsa bu işlemler sırasında kullnılabilir.

MIPS işlemcilerin sıfır durumuna göre dallanma yapması 32 bitlik bir register üzerinde 32 bitlik bir verinin sıfır olmasına yani 32 bitin ‘VE’ mantıksal birimiyle karşılaştırılmasına sebep olmaktadır. Bu işlem yerine bir bitin 0 yada 1 olduğunu kontrol etmek daha basit olacaktır.

8.1. Register Comparsion Unit (RCU)

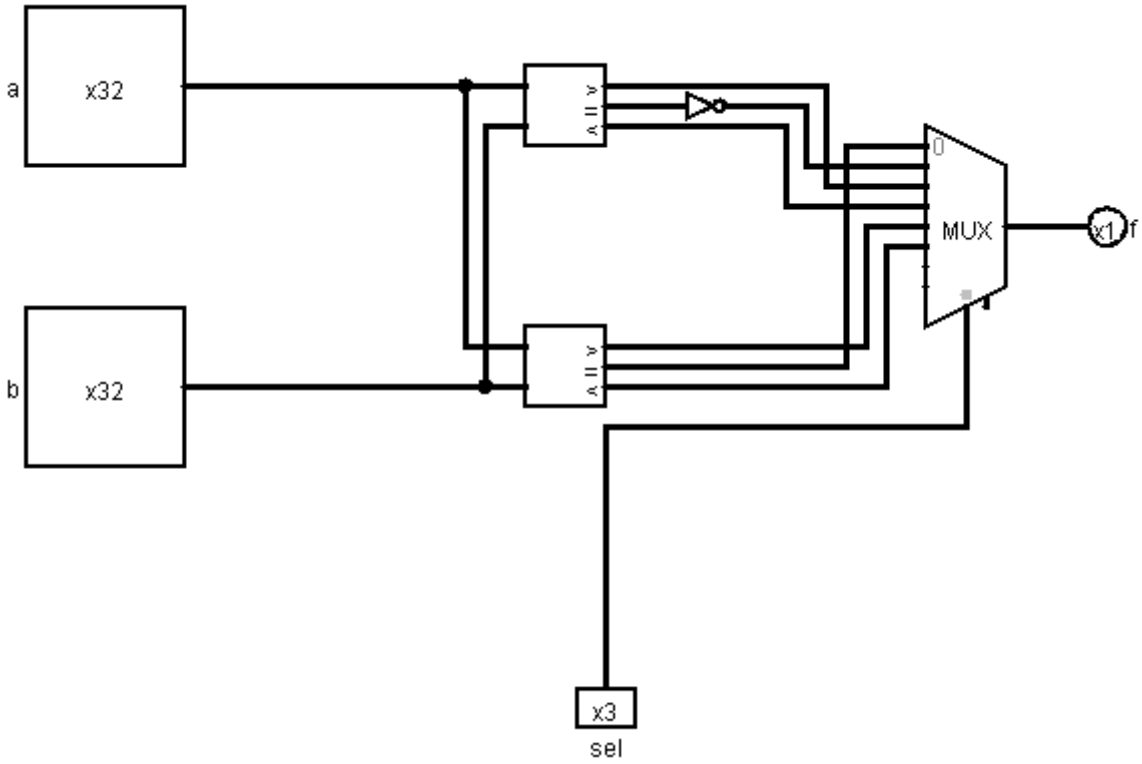
Bu birim işlemci çekirdeğine sonradan eklenecek birimlerdendir. Yapı olarak ALU ve BLU birimlerine benzemekte ve bu birimlerden farklı olarak RA biriminden veri alıp işlem sonucunu BRA birimine kaydetmek üzere tasarlanacaktır. İki register değeri için 32’şer bitlik iki giriş ve Bir bitlik sonuç çıktısı olacaktır. İçerisinde büyüktür, küçüktür, büyük eşittir, küçük eşittir, eşittir ve eşit değildir işlemlerini yapan devre yapısı bulunacak ve işlem seçimi 3 bitlik bir multiplexer ile sağlanacaktır. Yeni eklenecek dallanma işlemlerinde BRA üzerindeki bitlerin sıfır veya bir olmasına ayrı ayrı bakan iki komut olacağından dolayı

küçük eşittir, büyük eşittir ve eşit değildir işlemlerine net bir ihtiyaç bulunmamaktadır. Çünkü bu işlemler sırasıyla büyüktür, küçüktür ve eşittir işlemlerinin mantıksal tersidir. Ayrıca BLU biriminde ters alma işlemi dâhil olacağından dolayı direkt olarak bu sonuçların tersi alınabilir. Eğer bu işlemler çıkarılırsa karşılaştırma biriminde 2 bitlik bir multiplexer yeterli olacaktır.

MIPS komutlarında fonksiyon kısmı 6 bittir. Bu 2^6=64 farklı fonksiyonun tanımlanabilmesini sağlamaktadır. Karşılaştırma biriminde ihtiyaç olmadığı halde bu bitler aktif olarak kullanılabilecek durumdadır. Bu sebeple karşılaştırma işlemlerinin işlevselliğini arttırmak amacıyla işaretli ve işaretsiz olmak üzere karşılaştırma birimleri birbirinden ayrılmaya uygun durumdadır. Yukarıda verilen altı farklı işlem için altı adet işaretsiz işlem getirilebilir. Fakat eşittir ve eşit değildir işlemleri, işaretli ve işaretsiz işlemleri için her hangi bir farklılık göstermeyecektir. Bu yüzden bu iki işlemin bir defa tasarlanması yeterlidir. Ayrıca büyüktür işleminin tersi olan küçük eşittir, küçüktür işleminin tersi olan büyük eşittir işlemlerinin tanımlanma ihtiyacı bulunmamaktadır. Bu işlemlerin işaretli ve işaretsiz karşılaştırma işlemlerine katılmaması durumunda beş farklı işlem sistemin tam ihtiyacını görecek ve 3 bitlik bir multiplexer ile çalışmaya uygun hale gelecektir.

8.2. Register Comparsion Unit Tasarımı

Logisim uygulaması ile yapılmış olan tasarım Şekil 4'de gösterilmiştir. Bu tasarımda karşılaştırma işlemleri işaretli ve işaretsiz olarak yapılmaktadır.



Şekil 4. Register Comparsion Unit Logisim Tasarımı

Şekil 4'de logisim ile RCU biriminin tasarımı yapılmıştır. Bu birimin giriş ve çıkışları şöyledir:

- 32 bit birinci karşılştırma register değeri (a)
- 32 bit ikinci karşılaştırma register değeri (b)
- 3 bit işlem selektörü
- 1 bit karşılaştırma sonucu

Yukarıda Logisim uygulamasına ait iki karşılaştırma birimi kullanılmıştır. Çizimde aynı görüntüde olmalarına rağmen bunların temel farkı işaretli ve işaretsiz olarak farklı işlemler yapmalarıdır.

RCU biriminin tasarımı için yazılmış olan verilog kodları ise aşağıda bulunan kod bloğunda gösterilmiştir.

```
module cru(a,b,f,sel);
input[32:0] a,b;
input[2:0] sel;
output f;
reg f;
    always begin
        case (sel)
            3'b000:f<=(a==b);
            3'b001:f<=a!=b;
            3'b010:f<=a>b;
            3'b011:f<=a<b;
            3'b100:f<=$signed(a)>$signed(b);
            3'b101:f<=$signed(a)<$signed(b);
        endcase
    end
endmodule
```

9. MIPS İLE BİRLEŞTİRME

Çalışmanın nihayete ermesi amacıyla MIPS işlemci ile birleştirmenin nasıl olacağı bu bölümde anlatılacaktır. Öncelikle ek olarak oluşacak komutlarla komut entegrasyonu daha sonra yapılan birimlerin tümleşik devre gösterimi anlatılacaktır. Son olarak ise bu çalışmanın nihayete erdirilesi için derleyici tasarımı için öneriler anlatılacaktır. Derleyici tasarımının tamamen yapılmamasının sebebi ise bu konunun daha farklı bir çalışma alanı olmasıdır.

9.1. Ek Komutlar

Bu makale her ne kadar tüm işlemci çekirdeklerinde uygulanabilir bir mantık geliştirmiş olsa da bu makalede MIPS çekirdeği üzerinden çalışıldığından dolayı komutlar MIPS çekirdeğine göre planlanacak ve MIPS çekirdeğine uygun olarak düzenlenecektir.

32 bitlik MIPS çekirdeğinde bulunan temel komutlar 000000, 000010, 00011, 00100, 00101, 001000, 001001, 001010, 001011, 001100, 001101, 001111, 010000, 100011, 100100, 100101, 101000, 101001, 101011 bitleriyle başlamaktadır. Aşağıdaki tabloda bu komutların ilk dört bitlerine göre dağılımı gösterilmiştir.

Tablo 1. MIPS komutlarının opcode ve funct değerlerinin binary gösterimi

KOMUT	**OPCODE (6 bit)**	**FUNCT (6 bit)**
ADD	000000	100000
ADDU	000000	100001
AND	000000	100100
DIV	000000	011010
DIVU	000000	011011
JR	000000	001000
MFHI	000000	010000
MFLO	000000	010010
MULT	000000	011000
MULTU	000000	011001
NOR	000000	100111
XOR	000000	100110
OR	000000	100101
SLT	000000	101010
SLTU	000000	101011
SLL	000000	000000
SRL	000000	000010
SRA	000000	000011
SUB	000000	100010
SUBU	000000	100011
J	000010	000000
JAL	000011	000000
BEQ	000100	000000
BNE	000101	000000
ADDI	001000	000000
ADDIU	001001	000000
SLTI	001010	000000
SLTİU	001011	000000
ANDI	001100	000000
ORI	001101	000000
LUI	001111	000000
MFC0	010000	000000
LW	100011	000000
LBU	100100	000000
LHU	100101	000000
SB	101000	000000
SH	101001	000000
SW	101011	000000

MIPS çekirdeğinde bu komutlardan farklı olarak float point işlemlerine ait komutlar bulunmakta ve bunlar x1xx şeklinde değer alır. Dikkat edilmesi gereken ise float point haricindeki komutların x0xxxx mantığında (mfc hariç) float point komutlarının ise x1xxxx mantığında olmasıdır. Burada x ile ifade edilen ise 0 veya 1 değerlerinden herhangi birini alabilmesidir. Komut setinin ikinci sırasında bulunan bit komutun asıl çekirdek ile çalışıp çalışmadığını göstermektedir. Çünkü float point ünitesi farklı registerlar ile çalışıp farklı bir ALU ile işleme girmektedir.

Eklenecek yeni komutlar ana çekirdek ile çalışacağından dolayı x0xxxx mantığında olacak ve ana çekirdek dâhilinde işleme alınacaktır. Çünkü yeni eklenecek komutlar dallanma ve alt programı etkilecek ana çekirdeğin çalışmasını belirleyecektir. x0xxxx kullanımına bakıldığında 1011xx alanı boş bulunmaktadır. Yeni eklenecek komutlar xx değeri yerine 00, 01, 10, 11 olarak dağıtılabilir durumdadır. Dağıtım işlemi komutların çalışma prensibine göre yapılacaktır.

Eklenecek komutlar and.b ,or.b, not.b, xor.b, xnor.b, nand.b, nor.b,copy.b, lbit, sbit brh,brl, r.eq, r.neq, r.grnd, r.less, .r. grndu, r. lessu komutlarıdır. Bu komutlar içerisinde and.b ,or.b, not.b, xor.b, xnor.b, nand.b, nor.b, copy.b komutları bit lojik ünitesi ile çalışacağından dolayı opcode değerleri aynı olacaktır. Bu opcode değeri yukarıdaki tablo incelenirse ALU işlemlerinde 00 ile bitmekte olduğundan dolayı 00 ile bitmesi uygun bulunmaktadır. 00 ile bitmesi devresel bir kolaylık sağlamazken takip olarak kolaylık sağlayacaktır. Bu durumda bu komutların opcode değerleri 101100 şeklinde olacaktır. lbit, sbit komutları beş bitlik BRA ve ve beş bitlik RA adresi bulunduracağından dolayı 6 bitlik opcode değerlerinden sonraki 10 bit dolu olacaktır. Bu nedenle opcode değerleri farklı olmalıdır. 101110 ve 101111 değerleri verilebilir. 101101 değerinin verilmeme sebebi ise sondan ikinci bitlerini ayrı tutarak işlem kolaylığı sağlamaktır. Yani sondan 2. Biti doğal bir selektör olarak kullanmaktır. Bu durumda son boş kalan alan olan 101101 opcode alanına brh ve brl değerleri atanabilir.

Ek komutlar alanında ayrıca karşılaştırma komutları gerekmektedir. Bu komutlar MIPS komutları arasında bulunan bazı komutları işlevsiz hale getirecektir ve karşılaştırma işlemlerinin değerleri RA üzerinde sonucu ise BRA üzerinde saklanacağından dolayı daha farklı bir çalışma ile belirlenecektir. Burada karşılaştırma işlemleri için ALU ve BLU birimlerine paralel bir birim daha eklenmelidir. Bu birimi işleve sokan komutlar tek bir

opcode değeriyle çalışmalıdır. Tabloya baktığımız durumda 000010 (0x06) alanı boştur ve bu alan kullanılabilir. Karşılaştırma komutları tam liste olarak alınırsa r.eq, r.neq, r.grnd, r.less, .rgrndu, r.lessu olarak belirlenmiştir. Bu işlemler ALU ve BLU işlemleri gibi bir fonksiyon (funct) değeriyle birbirinden ayrılmaya müsaittir. Bu funct değerleri 000000, 000001, 000010, 000011, 000100, 000101 değerlerine sahip olacak ve v karşılaştırma birimi üzerinde 3 bitlik bir multiplexer'ı tetikleyecektir.

Tablo 2. MIPS komutlarının opcode ve funct değerlerinin binary gösterimi

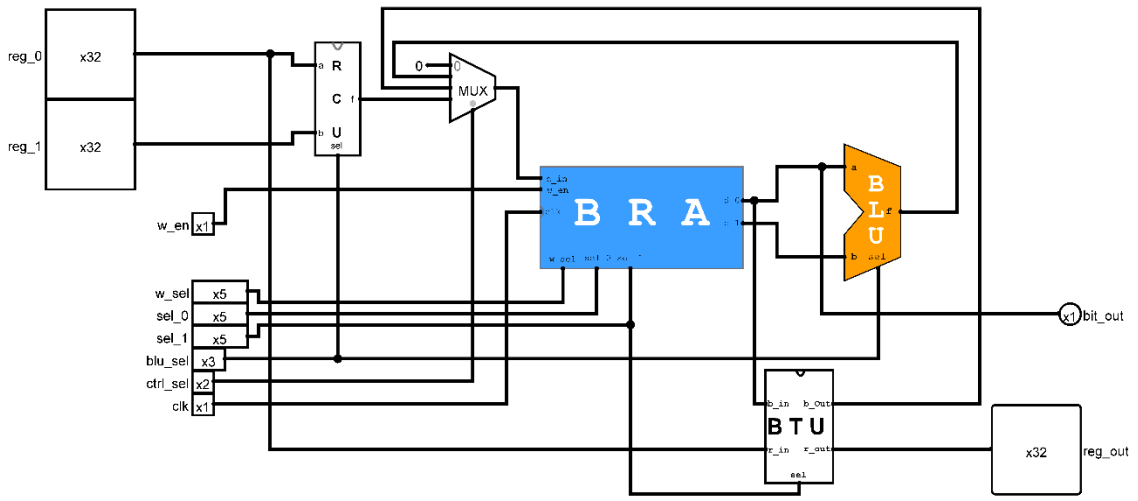
Komut	MIPS İçin Komut Tipi	Birim	Görev
and.b	R	BLU	Bitler arası and işlemi
or.b	R	BLU	Bitler arası or işlemi
nand.b	R	BLU	Bitler arası nand işlemi
nor.b	R	BLU	Bitler arası nor işlemi
not.b	R	BLU	Birinci bit not işlemi
xor.b	R	BLU	Bitler arası xor işlemi
xnor.b	R	BLU	Bitler arası xnor işlemi
copy.b	R	BLU	Bit değerinin transferi
lbit	R	BTU	Herhangi bir register üzerindeki herhangi bir bitlik veriyi BRA üzerine taşır.
sbit	R	BTU	BRA üzerindeki herhangi bir veriyi herhangi bir register üzerine taşır.
brh	I_2	-	Adres verilen bit 1 ise dallanma işlemi
brl	I_2	-	Adres verilen bit 0 ise dallanma işlemi
r.eq	R	RCU	register karşılaştır (eşit)
r.neq	R	RCU	register karşılaştır (eşit değil)
r.grnd	R	RCU	register karşılaştır (büyüktür)
r.less	R	RCU	register karşılaştır (küçüktür)
r.grndu	R	RCU	register karşılaştır (büyüktür işaretsiz)
r.lessu	R	RCU	register karşılaştır (küçüktür işaretsiz)

Yukarıdaki tabloda dikkat edilecek noktalardan birisi I_2 bir komut tipinin olduğudur. Normal şartlar altında MIPS mimarisinde böyle bir komut tipi bulunmamaktadır. I tipinde 2 farklı register adresi bulunmaktadır. 6 bit opcode, adet 5 bit register adresi ve 16 bit immed değer adresi bulunmaktadır. Fakat çalışma bazında dallanma işlemleri sadece BRA birimindeki bir bitlik bir adres ile belirlendiğinden bu birimlerde 6 bit opcode 5 bit adres ve 21 bit immed değeri alabilecektir. Bu nedenle bu formata çalışmada I_2 adı verilmiştir. 21 bit olmasının avantajı ise uzunca yazılmış yazılımlar için tek hmlede daha fazla kod parçacığında dallanma işlemi sağlayacaktır. Normal şartlarda 16 bitlik bir verinin dallanma için yetersizliği durumunda program başka bir dallanma noktasına giderek bu işlemi gerçekleştirebilecektir.

9.2. Birimlerim Tümleşik Devresi

Tasarlanan birimlerin sonuç olarak nasıl bir yapıya sahip olduğu ve nasıl çalışacağı mimari geliştirimi açısından önem arz etmektedir. Her ne kadar yapılacak tümleşik devre zorunlu olmasa da fikir vermesi açısından bu bölümde tasarlanacaktır. Bu tasarım Logisim uygulamasında yapılacak ve oluşan devrelere girişler ve çıkışlar kullanılarak nasıl hükmedileceği anlatılacaktır.

Logisim tasarımı aşağıdaki şekilde gösterilmiştir.



Şekil 5. Birimlerim Tümleşik Devresi Logisim Tasarımı

Girişler ve çıkışlar şöyledir:

- reg_0: Dışarıdan gelen register değeridir. Karşılaştırma işlemleri ve bit transferi için kullanılmaktadır.
- reg_1: Dışarıdan gelen ikinci register değeridir. Sadece karşılaştırma işlemlerinde kullanılır.
- w_en: BRA birimine yazmanın aktif olup olmayışını sağlar. Komuta göre işlemcinin çekirdeği kontrol eder.
- w_sel: yazma işleminin hangi bit değerini gösterdiğini adres olarak belirtir. BRA birimiyle çalışır.
- sel_0: BRA üzerinden ilk bit olarak hangi bitin okunacağını belirler
- sel_1: BRA biriminden ikinci bit olarak hangi bitin okunacağını belirler. Ayrıca BTU birimi için selektör görevindedir.
- blu_sel: BLU üzerinde hangi işlemin yapılacağını belirler. Yazılacak komutun funct bölümünden son 3 bit alınarak oluşturulur.
- ctrl_sel: Hangi işlemlerin aktif olduğunu belirler. Değeri devre üzerrinde bir mux ile kullanılır. 0 değeri gelirse 0 sonucunu, 1 değeri gelirse BLU işlem sonucu, 2 değeri gelirse BTU bit değeri ve 3 değeri gelirse RCU işlem sonucu verir.
- bit_out: sel_0 adresine göre BLU üzerindeki bit değerini verir. Dallanma işlemleri için çıkış verilmiştir.
- reg_out: BTU biriminde bir bitin register üzerindeki bir bite kaydedilmesi sonucu oluşan yeni register değerini göstermektedir.

9.3. Derleyici Tasarımı Hakkında

MIPS çekirdeğinde planlanan değişikliklerden sonra, bu değişikliklerin uygulanabilmesi açısından derleyici tasarımı gerekmektedir. Fakat bu çalışmada çalışma alanı dışında olduğu için derleyici tasarımı yapılmayacaktır. Genellikle işlemci mimarileri için temelde bir C derleyicisi tasarlansa da, bu işlemci için C derleyicisi yetersiz kalacaktır. Çünkü C dilinde boolean ifadeler kullanılmamakta ve karşılaştırma işlemleri nümerik değerler ile yapılmaktadır. Bu nedenle daha gelişmiş bir dil üzerinde yapılan derleyici çalışma açısından faydalı olacaktır. “stdbool.h” adında bir C kütüphanesi olmasına rağmen bu kütüphane sonradan yazılmış ve yazılan bir kod ile derlenmek üzerine hazırlanmıştır. [15] Bu çalışmanın aktivitesinin artması açısından derleyici yapılacak olan dilin boolean ifadeleri temelde yani işlemci mimarisinde çalıştıracak ve booleanı integer veya float gibi özel görerek derlemesi gerekmektedir.

C dilinin geliştirilmesi üzerine oluşturulmuş olan C++ dili boolean ifadeleri temelde derlemektedir. Bu sebeple yapılacak derleyici açısından uygun olacaktır. C++ için yapılacak derleyici sayesinde yeni eklenmiş olan birimler bu yazılım dilinde kendilerine yazılım dünyasında karşılık bulabileceklerdir. Her ne kadar C++ dili boolean değişkenleri direkt destekler konumda olsa da bu dil işlemci üzerinde boolean değişkenleri bit karşılık yerine daha büyük alanlarda kullanmaktadır. Bu bakımdan yeni eklenen birimlere uygun olarak çalışan bir derleyici C++ dilinin işlemci üzerinde performansını güçlendirecektir. Ayrıca boolean ifadelerin ayrılmasıyla birlikte assembly olarak sonuç alındığı takdirde, bu sonuçlar daha okunur olacaktır. Bu da derleyici veya işlemci mimarileri üzerine çalışmakta veya eğitim görmekte olan bireyler için faydalı ve makul olacaktır.

SONUÇ

Çalışmanın başından beri vurgulamış olduğumuz üzere bitler üzerinde işlem sağlayarak, boolean cebri işlemlerine destek vermek ve bu işlem sonuçlarıyla dallanma işlemlerini desteklemek assembly dillerinde kod okunurluğunu arttıracak ayrıca dallanma işlemlerinde mimarinin bit uzunluğu kadar bir veriyi kontrol etmeyip doğal yapısına uygun olarak bir bitlik verinin sonucuyla işlemlerin gerçekleşmesini sağlayacaktır. Boolean işlemleri MIPS mimarisinde olduğu gibi 32 bitlik bir alanda saklanmayacak, bunun yerine bir bitlik alanda saklanacaktır. Bu sayede 32 bitlik registr sayısı arttırılmadan, bir bitlik register kullanımı ile 32 bitlik register alanlarının boolean verilerden arındırılması böylece daha geniş bir register alanına sahip olması sağlanmış olacaktır.

Eklenen Bit Register Array birimi ile boolean işlemleri sonucu oluşan sonuçların saklanma alanı tanımlanmıştır. Bu alanda MIPS mimarisine göre çalışıldığında 32 tane 1 bitlik register alanı açılmıştır.

Bit Register Array biriminde bulunan bitler arasında işlem yapılıp sonuçlarını tekrardan bu birime kaydetmek için Bit Logical Unit adında bir birim oluşturulmuş ve çeşitli boolean cebrine ait işlemlerin burada yapılması öngörülmüştür. Bu sayede boolean işlemlerinin sadece 1 bitlik veriler ile yapılması planlanmıştır.

Bitlerin her ihtiyaç durumunda transfer edilebilmesi amacıyla Bit Transfer Unit isimli birim geliştirilmiş, bu birim register üzerinde bulunan bir bit değerini Bit Register Array birimine veya Bit register Array biriminde bulunan bir biti herhangi bir Register üzerine kopyalamayı sağlamıştır.

Son eklenen birim olarak Register Comparison Unit isimli birim ise register değerlerini işaretli veya işaretsiz olarak karşılaştırmayı sağlamış ve bu karşılaştırma sonucunu Bit register Array biriminde saklamak üzere planlanmıştır. Bu işlem sonuçları saklandıktan sonra istenildiği zaman kullanıma açılmıştır.

Toparlamak gerekirse bu dört birim sayesinde 1 bitlik olarak boolean işlemler işlemci çekirdeğinde bu birimler tarafından işlenebilir duruma getirilmiştir. Karşılaştırma sonucu oluşan bir bitlik sonuçlar da burada işlenmek üzere konumlandırılmıştır. Bu 1 bitlik veriler kullanılarak dallanma işlemlerinin bu veriler ile yapılması hedef olarak konulmuştur.

KAYNAKÇA

[1] C. O. Sanchez , «MiniMIPS: An 8-Bit MIPS in an FPGA for Educational Purposes,» %1 içinde *Reconfigurable Computing and FPGAs (ReConFig), 2011 International Conference on*, Cancun, Mexico, 2011.

[2] K. Yi ve Y.-H. Ding, «32-bit RISC CPU Based on MIPS Instruction Fetch Module Design,» %1 içinde *International Joint Conference on Artificial Intelligence*, Hainan Island, China, 2009.

[3] O. A. Siddiqui, R. Hasan, S. Mahmood ve A. R. Khan, «Simulators as a Teaching Aid for Computer Architecture and Organization,» %1 içinde *Intelligent Human-Machine Systems and Cybernetics (IHMSC)*, Nanchang, Jiangxi, China, 2012.

[4] V. Luković, R. Kmeta, A. Vulović, Đ. Damnjanović, A. Peulić, C. Dimopoulos ve K. Katzis, «Comparison of the effectiveness of Logisim software tool and remote experiments based on Nexys 2 FPGA platform in learning digital circuits design,» %1 içinde *Experiment@International Conference*, Faro, Portugal, 2017.

[5] G. Posser, G. Corrêa, R. Reis, L. Carro ve S. Bampi, «A MIPS-based ASIP to accelerate the inverse Hadamard tranform for H.264/AVC video coding,» %1 içinde *Circuits and Systems (LASCAS), 2010 First IEEE Latin American Symposium on*, Foz do Iguacu, Brazil, 2010.

[6] B. LE GAL ve C. JEGO, «DESIGN OF AN ASIP LDPC DECODER COMPLIANT WITH DIGITAL COMMUNICATION STANDARDS,» %1 içinde *Signal Processing Systems (SiPS), 2012 IEEE Workshop on*, Quebec City, QC, Canada, 2012.

[7] N. Saraswathi ve M. N. Topiwala, «Implementation of a 32-bit MIPS based RISC processor using Cadence,» %1 içinde *Advanced Communication Control and Computing Technologies (ICACCCT), 2014 International Conference on*, Ramanathapuram, India, 2014.

[8] H. Ma ve D. Wang, «The design of five-stage pipeline CPU based on MIPS,» %1 içinde *Electrical and Control Engineering (ICECE), 2011 International Conference on*, Yichang, China, 2011.

[9] H. Ma ve D. Wang, «The design of five-stage pipeline CPU based on MIPS,» %1 içinde *Electrical and Control Engineering (ICECE), 2011 International Conference on*, Yichang, China, 2011.

[10] T. Ball ve S. K. Rajamani, «Boolean Programs: A Model and Process For Software Analysis,» Microsoft Corporation, Redmond, WA, USA, 2000.

[11] http://www.cprover.org/satabs/, «Predicate Abstraction using SAT,» 02 01 2018.

[12] http://research.microsoft.com/slama, «Microsoft Slama,» 20 12 2017.

[13] R. Samanta, J. V. Deshmukh ve E. A. Emerson, «Automatic Generation of Local Repairs for Boolean Programs,» %1 içinde *Formal Methods in Computer-Aided Design, 2008. FMCAD '08*, Portland, OR, USA, 2008.

[14] https://www.cs.cornell.edu/courses/cs3410/2008fa/MIPS_Vol2.pdf, «MIPS32™ Architecture For Programmers,» 30 11 2016.

[15] https://en.wikibooks.org/wiki/C_Programming/stdbool.h, «C Programming/stdbool.h,» 10 10 2017.

ÖZGEÇMİŞ

2 Şubat 1990 tarihli Erzurum doğumluyum. İlköğretim ve lise eğitimimi Erzurumda tamamlamış olup Kütahya Dumlupınar Üniveristesi Bilgisayar Mühendisliği bölümünden mezun oldum. 2015 yılında Beykent Üniveristesi Fen Bilimleri Enstitüsü Bilgisayar Mühendisliği Bilim dalında yüksek lisans Eğitimime başladım.

Abubekir ŞAKAR