T.C.
SAKARYA UYGULAMALI BİLİMLER ÜNİVERSİTESİ TEKNOLOJİ FAKÜLTESİ
BİLGİSAYAR MÜHENDİSLİĞİ

MİKROİŞLEMCİLER
Ödevi

NİOS II – 32 BİT İŞLEMCİ TANITIMI

Adı Soyadı : Alikan KÖSE

Bölümü/Programı : Bilgisayar Mühendisliği

No : 23010903133

Nios II İşlemci ve Programlama Kılavuzu

İçindekiler

---

0. Giriş

Nios II Nedir?

Nios II, Altera (şu anda Intel FPGA) tarafından geliştirilen ve FPGA tabanlı sistemlerde kullanılmak üzere tasarlanmış bir gömülü işlemci çekirdeğidir. Nios II, kullanıcıların kendi donanım ve yazılım ihtiyaçlarına göre özelleştirilebilir bir işlemci mimarisi sunar. FPGA üzerinde çalıştığı için tasarımcılar, işlemcinin yapı taşlarını kendi projelerine özgü şekilde uyarlayabilirler. **FPGA (Field-Programmable Gate Array)**, donanım devrelerini programlayarak özelleştirme imkânı tanır. Bu esneklik sayesinde, Nios II işlemci çekirdeği, çevre birimlerini ve diğer donanım bileşenlerini ihtiyaca göre değiştirebilir, işlemci hızını, bellek yapısını ve komut setini projeye uygun olarak özelleştirebilir. Nios II, bu nedenle soft core bir işlemcidir.

Nios II Ailesi

Nios II işlemci ailesi üç ana varyanttan oluşur:

- **Nios II/f (Fast):** Yüksek performanslı uygulamalar için optimize edilmiştir.
- **Nios II/s (Standard):** Dengeli performans ve kaynak kullanımı sunar.
- **Nios II/e (Economy):** Düşük maliyetli ve kaynak sınırlı uygulamalar için uygundur.

Nios II işlemci, **Reduced Instruction Set Computer (RISC)** mimarisine sahip bir işlemcidir. Bu mimari, işlemcinin daha hızlı ve verimli çalışabilmesi için temel komutların basitleştirilmesini amaçlar. Nios II'nin aritmetik ve mantıksal işlemleri, genel amaçlı kayıtlar (registers) üzerinde gerçekleştirilmektedir. Bellek ile bu kayıtlar arasında veri hareketi, Load ve Store komutları ile sağlanır.

Nios II işlemcisinin kelime (word) uzunluğu 32 bit olup, tüm kayıtlar 32 bit uzunluğundadır. Bellek adreslemesi, little-endian veya big-endian tarzında yapılabilir ve bu stil, kullanıcı tarafından yapılandırma esnasında seçilebilir. Little-endian stilinde, kelimenin daha az anlamlı byte'ları (yani sağdaki byte'lar) düşük bellek adreslerine atanır.

Nios II, **Harvard mimarisi** olarak bilinen bir yapıyı kullanır. Bu, işlemcinin komut ve veri yollarının ayrı olduğu anlamına gelir, yani komutlar ve veri aynı anda farklı yollar üzerinden iletilir. Bu yapı, işlemcinin daha hızlı çalışmasına yardımcı olur. (Günümüz kişisel bilgisayarları, genel amaçlı kullanımlar için daha uygun olan **Von Neumann mimarisini** kullanır.)

Nios II işlemcisi üç farklı çalışma moduna sahiptir:

1. **Supervisor Modu:** Bu mod, işlemcinin tüm komutları çalıştırabilmesini ve tüm işlevleri yerine getirebilmesini sağlar. İşlemci sıfırlandığında bu modda başlar. Bu modda işlemci tüm işlem gücüne ve özelliklerine erişebilir.

2. **User Modu:** Bu mod, bazı komutların sadece sistem işleri için kullanılmasına izin verir ve bazı işlemci özelliklerine erişim sınırlıdır. Kullanıcı modunda, belirli komutlar veya sistem kaynakları sınırlıdır ve genellikle yalnızca uygulama kodları çalıştırılır. Nios II işlemcinin mevcut sürümleri, şu an için User modunu desteklememektedir.
3. **Debug Modu:** Yazılım hata ayıklama araçlarının breakpoint (duraklatma noktası) ve watchpoint (izleme noktası) gibi özellikleri uygulayabilmesi için kullanılan bir moddur. Debug modu, program geliştirilirken veya test edilirken çok yararlıdır.

1. Nios II Mimarisi

Register Set (Kayıt Kümesi)

Nios II işlemcisi, genel amaçlı ve özel amaçlı kayıtları içeren bir dizi kayıttan oluşur:

- **Genel Amaçlı Kayıtlar (General-Purpose Registers):** Nios II işlemcisi, 32 adet 32-bit genel amaçlı yazmaç (register) içerir. Bu yazmaçların bazıları özel amaçlar için ayrılmıştır ve Assembler (derleyici) tarafından tanınan özel adlara sahiptir. r0'dan r31'e kadar adlandırılmıştır.
- **Özel Amaçlı Kayıtlar (Special-Purpose Registers):** Program Sayacı (PC), Kontrol Kayıtları (Status, Exception, vb.)

1. **r0 – Sıfır Yazmacı (Zero Register):**
   - Tanım: Bu yazmaç her zaman 0 (sıfır) değerini içerir.
   - Okuma: Bu yazmaçtan bir değer okunduğunda sonuç her zaman 0 olur.
   - Yazma: Bu yazmaca bir değer yazılmaya çalışıldığında herhangi bir değişiklik olmaz, işlem yok sayılır.
   - Örnek `ADD r2, r0, r3 ;` : r3 değerini r2'ye kopyalar (çünkü sıfırdır ve toplama etkisizdir).

2. **r1 – Geçici Yazmaç (Temporary Register):**
   - Tanım: Assembler tarafından **geçici veri depolama** için kullanılır.
   - Kullanım: Kullanıcı programlarında doğrudan kullanılmamalıdır çünkü derleyici bu yazmacı geçici hesaplamalar için kullanır.

3. **r24 ve r29 – Özel Durum İşleme Yazmaçları (Exception Processing Registers):**
   - Tanım: Bu yazmaçlar **istisna durumlarını (exception)** işlerken kullanılır.
   - Kullanım: Kullanıcı modunda erişilemez, sadece işletim sistemi çekirdeği veya özel durum işleme sırasında kullanılabilir.

4. r27 ve r28 – Yığın Kontrol Yazmaçları (Stack Control Registers):

- Tanım: Bu yazmaçlar **yığın (stack)** kontrolü için kullanılır.
- **Yığın Nedir?** Programın çalışma zamanında dinamik olarak veri ve geri dönüş adreslerini tutan bir bellek bölgesidir.
- Kullanım: Fonksiyon çağrılarında ve geri dönüşlerinde yığının doğru çalışmasını sağlar.

5. r31 – Geri Dönüş Adresi Yazmacı (Return Address Register):

- Tanım: Bir alt program (**subroutine**) çağrıldığında, geri dönüş adresi bu yazmaca saklanır.
- Kullanım: Alt program tamamlandığında, işlemci bu adrese geri döner.
- Örnek

```
CALL subroutine  ; Alt program çağrılır ve geri dönüş adresi r31'e kaydedilir.

RET              ; r31'deki adrese geri dönülür.
```

Program Sayacı (PC)

Program Sayacı, bir sonraki çalıştırılacak talimatın adresini tutar. Kontrol talimatları (call, ret, jmp vb.) tarafından değiştirilir.

Kontrol Kayıtları

Nios II işlemcisi, işlemci durumunu ve kesme gibi özel durumları kontrol eden özel kontrol kayıtlarına sahiptir. Örneğin:

- **Status Register (Durum Kaydı):** Kesme durumları ve işlemci kipleri burada tutulur.
- **Exception Register (İstisna Kaydı):** İstisna nedenleri ve hata durumları kaydedilir.

Nios II işlemcisinde **6 adet 32-bit kontrol yazmacı** bulunur. Bu yazmaçlar **rdctl** (**read control**) ve **wrctl** (**write control**) komutları ile okunup yazılabilir.

Yazmaç	Görevi	Açıklama
ctl0	Durum Yazmacı	Mod ve kesme izin durumunu gösterir.
ctl1	İstisna Durumu Yazmacı	İstisna anındaki durum bilgilerini kaydeder.
ctl2	Hata Ayıklama Durumu	Hata ayıklama anındaki durum bilgilerini kaydeder.
ctl3	Kesme Etkinleştirme	Hangi kesmelerin etkin olduğunu belirler.
ctl4	Kesme Bekleme	Hangi kesmelerin aktif ve etkin olduğunu gösterir.
ctl5	İşlemci Kimliği	Çoklu işlemci sistemlerinde işlemciyi tanımlar.

2. Bellek ve G/Ç Aygıtlarına Erişim

Nios II işlemcisi, bellek ve G/Ç aygıtlarına erişim için esnek bir mimariye sahiptir. Bellek ve G/Ç cihazları doğrudan işlemci tarafından adreslenebilir ve bu adresleme, yükle (Load) ve sakla (Store) komutlarıyla gerçekleştirilir.

- **Bellek Önbelleği (Cache Memory):**
  Nios II işlemcisinin farklı sürümleri, önbellek kullanımında farklılık gösterir. Nios II/f (Fast) modeli, komut önbelleği ve veri önbelleği desteği sunar. Bu önbellekler FPGA üzerindeki bellek blokları kullanılarak oluşturulur ve boyutları SOPC Builder aracıyla belirlenir. Nios II/s (Standard) modeli sadece komut önbelleği içerirken, veri önbelleği desteği sunmaz. Nios II/e (Economy) modeli ise ne komut ne de veri önbelleği içerir, bu da onu daha düşük maliyetli ve kaynak dostu bir seçenek haline getirir.

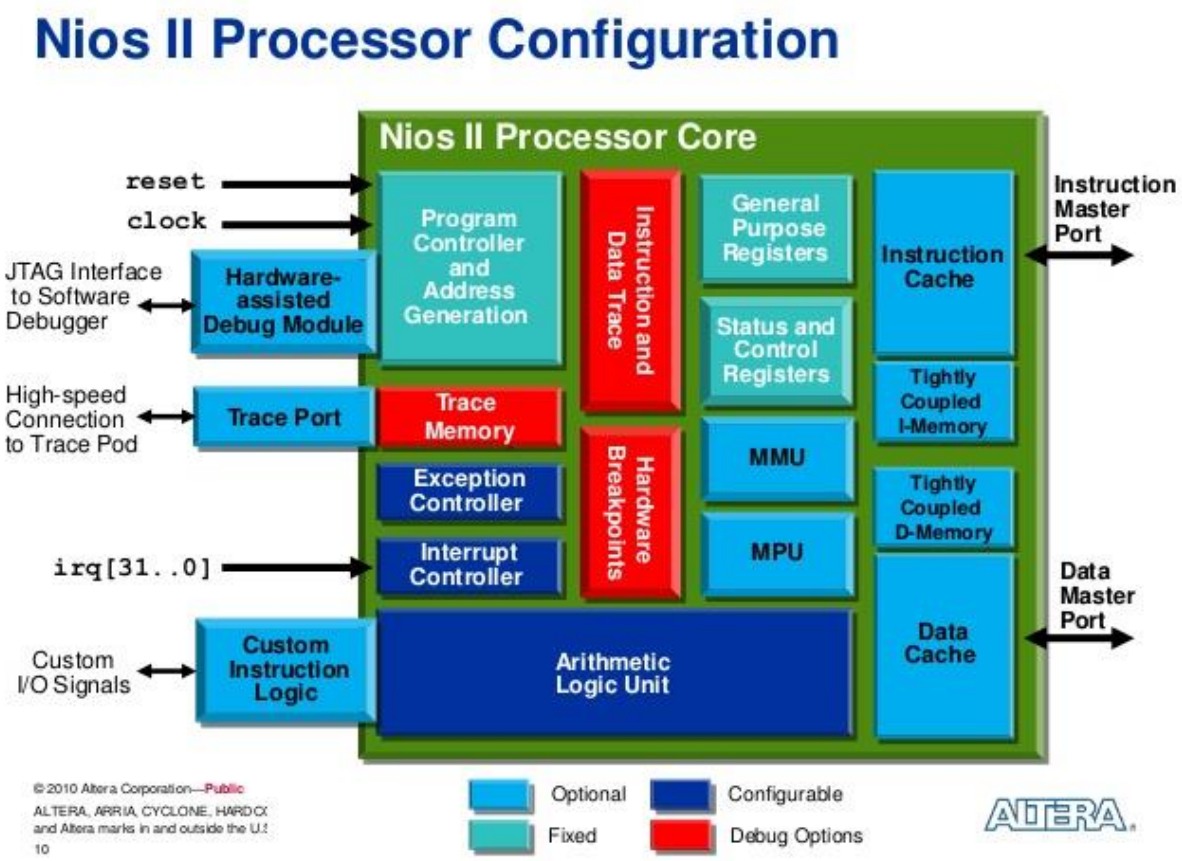
- **Sıkı Bağlı Bellek (Tightly Coupled Memory):**
  Sıkı bağlı bellekler, işlemciye doğrudan yol üzerinden bağlanır ve Avalon ağı üzerinden erişim yapmazlar. Bu bellekler, önbelleği bypass ederek daha hızlı erişim sağlar ve gerçek zamanlı işlemler için idealdir. Eğer sistemde komut önbelleği yoksa, Nios II/f ve Nios II/s işlemciler için en az bir sıkı bağlı bellek sağlanmalıdır. Birden fazla sıkı bağlı komut ve veri belleği yapılandırılabilir.

- **FPGA Dahili Bellek (On-Chip Memory):**
  FPGA üzerindeki dahili bellek bloklarına erişim, Avalon Network veya Tightly Coupled Memory üzerinden yapılabilir. Dahili bellek, işlemci ve diğer bileşenler arasında hızlı veri paylaşımı sağlar.

- **Harici Bellek (Off-Chip Memory):**
  Harici bellekler, işlemciye uygun arabirimler aracılığıyla bağlanır. SRAM hızlı erişim sağlarken, SDRAM büyük miktarda veri depolamak için kullanılır. Flash bellek ise kalıcı veri saklama için uygundur. SOPC Builder aracıyla uygun arabirim modülleri eklenerek bağlantı sağlanır.

- **G/Ç Aygıtlarına Erişim (I/O Access):**
  Nios II işlemcisi, Memory-Mapped I/O (Bellek Eşlemeli G/Ç) yöntemini kullanır. G/Ç aygıtları, işlemcinin adres alanında bellek bölgeleri gibi görülür ve her G/Ç cihazının bir bellek adresi vardır. Bu adresler üzerinden veri alışverişi, yükle (Load) ve sakla (Store) komutlarıyla gerçekleştirilir.

```
ldw r2, 0x80001000  ; Adres 0x80001000'den veriyi oku ve r2 yazmacına yükle.

stw r3, 0x80001004  ; r3 yazmacındaki veriyi adres 0x80001004'e yaz.
```



- **Nios II İşlemcisinde Bellek Yönetimi ve Koruması :**
  Nios II işlemcisinde **MMU (Memory Management Unit)** ve **MPU (Memory Protection Unit)**, bellek erişimini ve güvenliğini sağlayan iki önemli bileşendir. **MMU**, sanal bellek adreslerini fiziksel adreslere dönüştürerek gelişmiş bellek yönetimi yapar ve büyük, karmaşık sistemlerde kullanılır. Sanal bellek, sayfa tablosu gibi yapılarla yönetilir ve işlemciye belleği daha esnek bir şekilde kullanma imkanı sunar. **MPU** ise daha basit bir yapı sunar ve bellek alanlarına erişim kontrolü sağlar, bu da belleği korumak ve hatalı erişimleri engellemek için kullanılır. MPU, genellikle daha küçük, güvenli sistemlerde tercih edilir, çünkü yalnızca bellek erişim izinlerini denetler ve sanal bellek işlemleri gerçekleştirmez. Her iki birim de Nios II'yi, belleği daha güvenli ve verimli kullanabilmek için özelleştirme imkanı sunar.

Ek Bilgiler

- **Bellek Haritalama ve Belirleme (Memory Mapping and Addressing):** Nios II işlemcisi, bellek ve G/Ç aygıtlarını belirli adres aralıklarına haritalar. Bu, sistem tasarımında esneklik sağlar ve işlemcinin farklı bellek türlerine ve G/Ç cihazlarına hızlı erişimini mümkün kılar. SOPC Builder veya Platform Designer araçları kullanılarak bu haritalama yapılabilir.
- **Kesme Yönetimi (Interrupt Management):** Nios II işlemcisi, kesme (interrupt) mekanizması ile G/Ç cihazlarından gelen olayları işleyebilir. Kesme kontrol birimi (Interrupt Controller), önceliklendirme ve kesme işleme işlevlerini yönetir.
- **Bellek Erişim Hızı ve Verimlilik:** Bellek erişim hızını artırmak için sıkı bağlı bellek ve önbelleklerin doğru yapılandırılması önemlidir. Ayrıca, bellek hiyerarşisinin (önbellek, sıkı bağlı bellek, dahili ve harici bellek) etkin kullanımı, sistem performansını optimize eder.

3. Adresleme Modları

Nios II işlemcisi, 32-bit adresler kullanır. 32-bit adresleme, teorik olarak 4 GB bellek alanına erişim sağlar. Bellek, **8-bit (1 bayt)** birimler halinde adreslenir. Nios II, 32-bit (kelime), 16-bit (yarım kelime) veya 8-bit (byte) veri okuma ve yazma işlemleri gerçekleştirebilir. Mevcut olmayan bir bellek veya G/Ç konumuna okuma veya yazma işlemi gerçekleştirildiğinde, sonuç tanımsız olur. Nios II işlemcisi tarafından sağlanan beş farklı adresleme modu mevcuttur:

1. **Immediate Mode (Anında Değer Modu):**
   İşlemci komutunda 16-bitlik bir sabit değer doğrudan belirtilir. Aritmetik işlemlerde, 16-bit değer **işaret (sign-extended)** genişletilerek **32-bit** hale getirilir.

```
addi r2, r3, 100  ; r3 yazmacındaki değere 100 ekle ve
sonucu r2 yazmacına kaydet.
```

2. **Register Mode (Yazmaç Modu) :**
   Operant (veri) **doğrudan bir yazmaçta** (register) bulunur. İşlemler, doğrudan yazmaçtaki değer üzerinde yapılır. Hızlı veri erişimi sağlar.

```
add r4, r5, r6  ; r5 ve r6 yazmaçlarındaki değerleri topla
ve sonucu r4 yazmacına yaz.
```

3. Displacement Mode (Yer Değiştirme Modu):

Etkili adres, bir yazmaç içeriği ile komut içerisinde belirtilen 16-bit imzalı bir **yer değiştirme (displacement)** değerinin toplamıdır. Dizilere veya yapıların belirli alanlarına erişimde kullanılır.

```
ldw r2, 16(r3)  ; r3 yazmacındaki adrese 16 bayt ekle ve o
adresteki kelimeyi (32-bit) r2 yazmacına yükle.
```

4. **Register Indirect Mode (Dolaylı Yazmaç Modu):**

Etkili adres, bir yazmacın içeriği ile belirlenir. Göstericiler (pointers) ile çalışmak için uygundur.

```
ldw r2, (r3)  ; r3 yazmacındaki adresi oku ve o adresteki
kelimeyi r2 yazmacına yükle.
```

5. **Absolute Mode (Mutlak Adresleme Modu):**

Displacement modu kullanılarak, **r0 yazmacı (her zaman 0 değerine sahiptir)** ile mutlak bir adres hesaplanır. Bellekte sabit bir adrese doğrudan erişim sağlamak için kullanılır.

```
ldw r2, 0x1000(r0)  ; Mutlak adres 0x1000'den veriyi oku ve r2
yazmacına yükle.
```

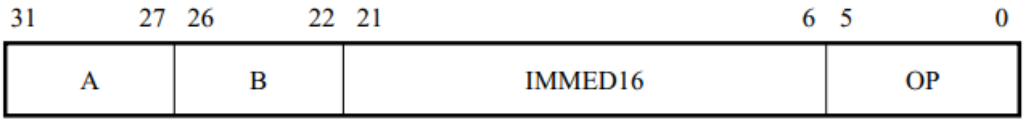
4. Komut Formatları

Nios II işlemcisi, 32-bit uzunluğunda komutlar kullanır ve bu komutlar üç ana formatta sınıflandırılabilir: **I-type (Immediate-type), R-type (Register-type)** ve **J-type (Jump-type).** Bu formatlar, işlemciye hangi verilerin, hangi kaydediciler üzerinden işleneceğini ve komutların nasıl kodlandığını belirler. Her formatın kendine özgü kullanım amacı ve komut yapısı vardır.

1. **I-type (Immediate Type) :**

Bu format, özellikle doğrudan immediate değerler ile yapılan işlemlerde kullanılır.

Yapısı:



(a) I-type

- b31-b27: A kaydedici (5-bit) - İlk kaydedici.
- b26-b22: B kaydedici (5-bit) - İkinci kaydedici.
- b21-b6: IMMED16 (16-bit) - Sabit (immediate) değeri, işlemciye iletilen sayıdır. Bu değerin işleme dahil edilmesinden önce işareti uzatılabilir (sign-extend).
- b5-b0: Op kod (6-bit) - Komut türünü belirler.

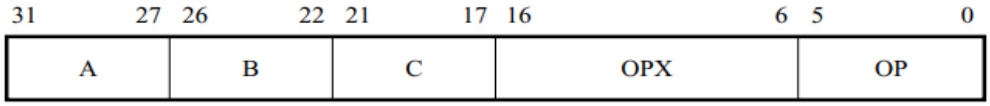
```
addi r2, r3, 100  ; r3 yazmacındaki değere 100 ekle ve sonucu
r2 yazmacına kaydet.

addi r2, r3, 0x64  ; r3 yazmacındaki değere 0x64 (hexadecimal -
100 decimal) ekle ve sonucu r2 yazmacına kaydet.
```

Not : **Immediate değerler, decimal veya hexadecimal formatlarda belirtilebilir.** Decimal formatta belirtmek için normal rakamlar kullanılırken, hexadecimal formatta belirtmek için 0x öneki kullanılır. Hangi formatı kullanacağınız, okuyucunun anlayışına ve kodun okunabilirliğine bağlı olarak seçilebilir.

2. **R-type (Register Type):**

Bu format, yalnızca **kaydediciler arasında** yapılan işlemler içindir. Örneğin, bir kaydedicinin içeriği başka bir kaydediciye aktarılabilir veya kaydediciler arasında aritmetik işlemler yapılabilir.

Yapısı:



(b) R-type

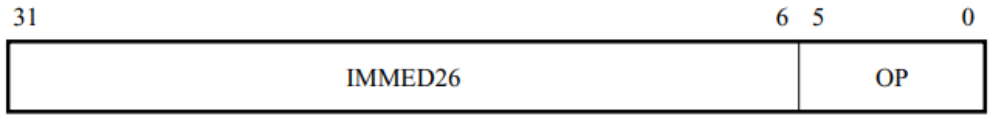
- b31-b27: A kaydedici (5-bit) - İlk kaydedici.
- b26-b22: B kaydedici (5-bit) - İkinci kaydedici.
- b21-b17: Hedef C kaydedici, işlemin sonucu burada saklanır.
- b16-b6: OPX (11-bit) - Op kodunu uzatır (ekstra işlem belirleme).
- b5-b0: Op kod (6-bit) - Komut türünü belirler.

```
add r1, r2, r3  ; r2 ve r3'ü topla, sonucu r1'e yaz

sub r2, r3, r4  ; r3 yazmacındaki değerden r4 yazmacındaki
değeri çıkar ve sonucu r2 yazmacına kaydet.
```

Not : Nios II işlemcisinde, bazı komutların daha spesifik işlemlerini belirlemek için temel opcode alanı yeterli olmaz. Bu durumda, **OPX (Extended Opcode)** alanı kullanılarak opcode uzatılır. OPX, 11 bitlik bir alan olup, komutların daha fazla varyantını ve spesifik işlevlerini tanımlamak için kullanılır. Bu sayede, komut seti genişletilebilir ve işlemcinin esnekliği artırılır. Temel opcode komutun genel işlevini belirlerken, OPX alanı bu işlevin daha detaylı ve spesifik varyantlarını belirler. Bu, daha karmaşık ve çeşitli komutların desteklenmesini sağlar.

3. **J-type (Jump Type):**
Bu format, genellikle **program sırasını değiştiren komutlar** (örneğin, **jump** veya **call**) için kullanılır. Komut, belirli bir **adres**i belirtmek için 26-bitlik bir sabit (immediate) içerir.

Yapısı:



(c) J-type

- b31-b6: 26-bitlik **IMMED26** alanı - Atlama adresi veya işlemciye yapılacak çağrı için adres bilgisi içerir.
- b5-b0: Op kod (6-bit) - Komut türünü belirler.

```
br 1000  ; Adres 1000'e dallanma yapar.
```

Not : Nios II işlemcisinde J-type (Jump Type) komutları, 26-bitlik bir (immediate) değeri içerir. Bu sabit değer, atlama adresini belirtir. Bellek adreslerinin 32-bit olmasına rağmen 26-bitlik bir alan kullanılır çünkü adresin düşük 2 biti her zaman 0'dır (her komut 4 byte uzunluğundadır). Bu 26-bitlik immediate değer, program sayacının (PC) en yüksek bitleri ile birleştirilerek tam 32-bitlik bir adres oluşturulur. Bu sayede, işlemci büyük bellek alanlarını etkili bir şekilde adresleyebilir.

5. Load ve Store Komutları

Load ve **Store** komutları, bellek (veya I/O arabirimleri) ile genel amaçlı kaydediciler arasında veri taşıma işlemlerini gerçekleştirir. Bu komutlar **I-type** formatına sahiptir

- **Load Word (ldw) Komutu :**

ldw rB, byte_offset(rA) komutu, bellek adresini hesaplamak için **rA** kaydedicisinin içeriği ile belirtilen **byte_offset** değerinin toplamını kullanır. 16-bit olan **byte_offset** değeri, 32-bit olarak işaret genişletilir. Bu hesaplanan adres, bellekten veri alır ve bu veri **rB** kaydedicisine yüklenir.

```
ldw r2, 4(r1) ; Bu komut, r1 kaydedicisindeki değeri 4 ile
toplayarak bellek adresini bulur. Ardından, bu adresin içeriğini r2
kaydedicisine yükler. Yani, r1 + 4 adresindeki 32-bit veri r2
kaydedicisine aktarılır.
```

- **Store Word (stw) Komutu :**

stw rB, byte_offset(rA) komutu, **rB** kaydedicisinin içeriğini bellek adresine yazar. Bu adres, **rA** kaydedicisinin içeriği ile belirtilen **byte_offset** değerinin toplamına dayanır.

```
stw r3, 8(r4) ; Bu komut, r3 kaydedicisinin içeriğini, r4
kaydedicisindeki değeri 8 ile toplayarak bulunan bellek adresine
yazar. Yani, r4 + 8 adresine r3 kaydedicisinin içeriği yazılır.
```

Load ve Store Komutları (Daha Kısa Operanlar İçin):

Bazı durumlarda daha kısa veri türleri (8-bit veya 16-bit) kullanılır. Bu tür verilerle yapılan **Load** ve **Store** işlemleri için şu komutlar bulunur:

- **Load Byte**: ldb (8-bit veri yükler)
- **Load Byte Unsigned**: ldbu (8-bit veri yükler, işaretsiz)
- **Load Halfword**: ldh (16-bit veri yükler)
- **Load Halfword Unsigned**: ldhu (16-bit veri yükler, işaretsiz)

Yükleme işleminden sonra, 8-bit veya 16-bit veriler, 32-bit kaydediciye sığdırılacak şekilde **sign extend** (işaret genişletme) veya **zero extend** (sıfır genişletme) yapılır.

- **ldb** ve **ldh** komutları işaret genişletme kullanır.
- **ldbu** ve **ldhu** komutları sıfır genişletme kullanır.

Store Komutları (Daha Kısa Operanlar İçin):

- **Store Byte:** stb (Kaydedici B'nin düşük byte'ını bellek adresine yazar)
- **Store Halfword:** sth (Kaydedici B'nin düşük yarım kelimesini bellek adresine yazar)

sth komutunda, hedef bellek adresi **yarım kelime hizasında** (halfword aligned) olmalıdır. Yani, adresin 2'ye bölünmesiyle kalanı sıfır olmalıdır.

I/O Bellek Erişim Komutları:

I/O cihazlarıyla etkileşim için özel **Load** ve **Store** komutları da vardır. Bu komutlar, belleğe yapılan normal erişimlerden farklı olarak, varsa **cache** belleği atlar. Bu komutlar şunlardır:

- **Load Word I/O:** ldwio
- **Load Byte I/O:** ldbio
- **Load Byte Unsigned I/O:** ldbuio
- **Load Halfword I/O:** ldhio
- **Load Halfword Unsigned I/O:** ldhuio
- **Store Word I/O:** stwio
- **Store Byte I/O:** stbio
- **Store Halfword I/O:** sthio

Bu komutlar, **I/O cihazlarının bellek adreslerine** veri taşır ve herhangi bir önbelleği atlar (cache bypass).

- **Doğrudan Adres Kullanımı (Absolute Addressing)**

  Örneğin 3000. adrese doğrudan veri yazmak için, işlemci kaydedicilerinin birini kullanarak adresi hesaplamak ve bu adrese veri yazmak gerekecek. Ancak, Nios II'nin **store** komutları genellikle kaydediciler ve ofset kullanılarak yapılır. Bu durumda, adresi 0'dan hesaplamak için, örneğin r0'ı (her zaman sıfır olan kayıt) kullanabiliriz.

```
    li   r2, 1000          ; r2 kaydedicisine 1000 değerini yükle
    li   r1, 3000          ; r1 kaydedicisine 3000 adresini yükle
    stw  r2, 0(r1)         ; r2'deki değeri, r1'deki adrese (3000)
```

6. Aritmetik İşlem Komutları (Arithmetic Instructions)

Arithmetik talimatlar, genel amaçlı kaydedicilerdeki veriler veya talimatlarda verilen anlık değerler üzerinde işlem yapar. Bu talimatlar, genellikle **R-tip** veya **I-tip** formatlarında olur. İşte bazı temel aritmetik talimatlar ve nasıl çalıştıkları:

1. **Add (Toplama) Talimatı:**
   - **add rC, rA, rB:** Bu talimat, **rA** kaydedicisindeki değeri **rB** kaydedicisindeki değerle toplar ve sonucu **rC** kaydedicisine yazar. Bu işlem, iki kaydedicinin içeriklerinin toplamını hesaplar.
     - Örnek: add r3, r1, r2 → **r1** + **r2** → Sonuç **r3**'te saklanır.
2. **Add Immediate (Anlık Toplama) Talimatı:**
   - **addi rB, rA, IMMED16:** Bu talimat, **rA** kaydedicisindeki değeri, talimatta verilen **16-bit** işaretli **IMMED16** değeri ile toplar ve sonucu **rB** kaydedicisine yazar. Burada **IMMED16** sayısı işaretli bir değer olduğundan, talimatta verilen değerin işaret uzatılması (sign extension) yapılır.
     - Örnek: addi r2, r1, 5 → **r1** + 5 → Sonuç **r2**'de saklanır.
3. **Sub (Çıkarma) Talimatı:**
   - **sub rC, rA, rB:** Bu talimat, **rB** kaydedicisindeki değeri **rA** kaydedicisindeki değerden çıkarır ve sonucu **rC** kaydedicisine yazar.
     - Örnek: sub r3, r1, r2 → **r1** - **r2** → Sonuç **r3**'te saklanır.
4. **Sub Immediate (Anlık Çıkarma) Talimatı:**
   - **subi:** Bu aslında bir takma talimattır ve **addi rB, rA, -IMMED16** şeklinde uygulanır. **rA** kaydedicisindeki değerden **IMMED16** değerini çıkarır ve sonucu **rB** kaydedicisine yazar.
     - Örnek: subi r2, r1, 5 → **r1** - 5 → Sonuç **r2**'de saklanır.
5. **Multiply (Çarpma) Talimatı:**
   - **mul rC, rA, rB:** Bu talimat, **rA** ve **rB** kaydedicilerindeki değerleri çarpar ve çarpımın düşük 32-bit'lik kısmını **rC** kaydedicisine yazar. Bu işlemdeki sayılar işaretsiz (unsigned) sayılar olarak ele alınır.
     - Örnek: mul r3, r1, r2 → **r1** * **r2** → Sonuç **r3**'te saklanır.
   - **muli rB, rA, IMMED16:** Bu, **mültiplikasyonun** anlık (immediate) versiyonudur. **rA** kaydedicisindeki değeri, işaretli bir şekilde uzatılmış **IMMED16** değeriyle çarpar ve sonucu **rB** kaydedicisine yazar.
     - Örnek: muli r2, r1, 10 → **r1** * 10 → Sonuç **r2**'de saklanır.

6. Divide (Bölme) Talimatı:

- **div rC, rA, rB:** Bu talimat, **rA** kaydedicisindeki değeri, **rB** kaydedicisindeki değere böler ve bölümün tam sayı kısmını **rC** kaydedicisine yazar. Burada, operandlar işaretli (signed) sayılar olarak kabul edilir.
    - Örnek: div r3, r1, r2 → **r1** / **r2** → Sonuç **r3**'te saklanır.
- **divu rC, rA, rB:** Bu talimat, **rA** kaydedicisindeki değeri, **rB** kaydedicisindeki değere böler, ancak burada operandlar işaretsiz (unsigned) sayılar olarak kabul edilir.
    - Örnek: divu r3, r1, r2 → **r1** / **r2** (işaretsiz) → Sonuç **r3**'te saklanır.

Notlar:

- **Taşma ve Taşıma:** Bu aritmetik işlemler taşma (overflow) ve taşıma (carry) durumlarını kontrol etmezler. Eğer taşma veya taşıma kontrolü yapmak gerekiyorsa, ek talimatlar (örneğin, addc, subc) kullanılır.
- **İşaretli ve İşaretsiz Sayılar:** Çarpma ve bölme işlemlerinin işaretli ve işaretsiz sayılarla nasıl yapıldığını bilmek önemlidir. **mul** ve **div** işaretli sayılarla çalışırken, **mul** ve **divu** işaretsiz sayılarla çalışır.

7. Mantıksal İşlem Komutları (Logic Instructions)

Nios II işlemcisindeki **Logic Instructions** (Mantıksal Talimatlar), mantıksal **AND**, **OR**, **XOR** ve **NOR** işlemlerini gerçekleştiren talimatlardır. Bu işlemler, genellikle veri üzerinde bit düzeyinde mantıksal işlemler yapar. Mantıksal talimatlar, hem genel amaçlı kayıtlarda bulunan verilere hem de talimatta verilen 16-bitlik **immediate** (doğrudan değer) verilere uygulanabilir.

1. AND Talimatı:

and rC, rA, rB

Bu talimat, **register A** ve **register B**'deki verileri bit düzeyinde **AND** işlemi yaparak **register C**'ye kaydeder.

Örneğin:

```
rA = 0b1100 ;   register A'ya 1100 binary verisi
rB = 0b1010 ;   register B'ye 1010 binary verisi
and rC, rA, rB ;  rC = 0b1000   (bitwise AND işlemi sonucu)
```

2. OR Talimatı:

or rC, rA, rB

Bu talimat, **register A** ve **register B**'deki verileri bit düzeyinde **OR** işlemi yaparak **register C**'ye kaydeder.

Örneğin:

```
rA = 0b1100   // register A'ya 1100 binary verisi
rB = 0b1010   // register B'ye 1010 binary verisi
or rC, rA, rB // rC = 0b1110   (bitwise OR işlemi sonucu)
```

3. XOR Talimatı:

xor rC, rA, rB

Bu talimat, **register A** ve **register B**'deki verileri bit düzeyinde **XOR** işlemi yaparak **register C**'ye kaydeder.

Örneğin:

```
rA = 0b1100   // register A'ya 1100 binary verisi
rB = 0b1010   // register B'ye 1010 binary verisi
xor rC, rA, rB // rC = 0b0110   (bitwise XOR işlemi sonucu)
```

4. NOR Talimatı:

nor rC, rA, rB

Bu talimat, **register A** ve **register B**'deki verileri bit düzeyinde **NOR** işlemi yaparak **register C**'ye kaydeder. NOR, OR işleminden sonra tüm bitlerin tersini alır.

Örneğin:

```
rA = 0b1100   // register A'ya 1100 binary verisi
rB = 0b1010   // register B'ye 1010 binary verisi
nor rC, rA, rB // rC = 0b0001   (bitwise NOR işlemi sonucu)
```

16-Bit Immediate (Doğrudan Değer) ile Mantıksal Talimatlar

Mantıksal talimatlarda, **16-bit immediate** (doğrudan değer) kullanarak işlemler yapılabilir. Bu durumda, 16-bitlik değer 32-bit'e sıfırla uzatılır (zero-extended). Bu tür talimatlar, genellikle bir kaydın tüm bitleriyle işlem yapmak yerine sadece üst 16 bitini hedefler.

- **AND Immediate (İmmidiate AND):**

  andi rB, rA, IMMED16

  Bu talimat, **register A** ve verilen **IMMED16** (16-bit immediate) değerini bit düzeyinde **AND** işlemi yaparak **register B**'ye kaydeder.

```
rA = 0b1111111111111111   // register A'ya 16-bit değer ( Bu şekilde veri atanamaz, veri sadece hexadecimal ve decimal olarak atılabilir. Burada registerin içeriğinin kolay anlaşılabilir olması için bu şekilde belirtilmiştir.)

andi rB, rA, 0xF0F0        // immediate değeri 0xF0F0

// rB = 0b1111000011110000   (AND işlemi sonucu)
```

- **OR Immediate (İmmidiate OR) :**

  ori rB, rA, IMMED16

  Bu talimat, **register A** ve verilen **IMMED16** (16-bit immediate) değerini bit düzeyinde **OR** işlemi yaparak **register B**'ye kaydeder.

- **XOR Immediate (İmmidiate XOR):**

  xori rB, rA, IMMED16

  Bu talimat, **register A** ve verilen **IMMED16** (16-bit immediate) değerini bit düzeyinde **XOR** işlemi yaparak **register B**'ye kaydeder.

- **OR Immediate (İmmidiate OR) :**

  ori rB, rA, IMMED16

  Bu talimat, **register A** ve verilen **IMMED16** (16-bit immediate) değerini bit düzeyinde **OR** işlemi yaparak **register B**'ye kaydeder.

  Nios II işlemcisinde **nori (nor immediate)** komutu bulunmamaktadır.

8. Taşıma Komutları (Move Instructions)

Nios II işlemcisinde **Move Instructions** (Taşıma Talimatları), bir kaydın içeriğini başka bir kayda kopyalamak veya bir **immediate** değeri bir kayda yüklemek için kullanılır. Bu talimatlar, genellikle diğer talimatlar kullanılarak implementasyon edilir ve **psödotalimatlar** olarak adlandırılır.

1. Mov (Taşıma Talimatı):

mov rC, rA

Bu talimat, **register A**'daki veriyi **register C**'ye kopyalar. **Nios II** işlemcisinde bu talimat, aslında bir **add** talimatı olarak implement edilir:

- add rC, rA, r0

Burada **r0**'ın değeri her zaman sıfırdır (zero register). Bu nedenle, **register A**'daki veri **register C**'ye kopyalanmış olur.

2. Move Immediate (Doğrudan Değer Taşıma):

movi rB, IMMED16

Bu talimat, verilen **16-bit** IMMED16 değerini **sign extension** ile **32-bit**'e uzatarak **register B**'ye yükler. Yani, işaretli genişletme ile negatif değerler doğru şekilde işlenir. Bu talimat aslında şu şekilde implement edilir:

- addi rB, r0, IMMED16

Burada, **r0**'ın değeri sıfır olduğu için **IMMED16** doğrudan **register B**'ye yüklenir.

3. Move Unsigned Immediate (İşaretsiz Doğrudan Değer Taşıma):

movui rB, IMMED16

Bu talimat, verilen **16-bit** IMMED16 değerini **zero extension** ile **32-bit**'e uzatarak **register B**'ye yükler. Burada **zero extension** ile, negatif sayılar sıfırla genişletilir. Bu talimat, şu şekilde implement edilir:

- ori rB, r0, IMMED16

Bu işlemin sonucu, **IMMED16** değeri **register B**'ye sıfırla genişletilmiş şekilde yüklenir.

Özetle:

- **mov:** Bir kaydın içeriğini başka bir kayda kopyalar, aslında **add** talimatı olarak implement edilir.
- **movi:** Bir **16-bit immediate** değeri **32-bit**'e işaretli genişletme (sign extension) ile yükler, aslında **addi** talimatı ile implement edilir.
- **movui:** Bir **16-bit immediate** değeri **32-bit**'e sıfırla genişletme (zero extension) ile yükler, aslında **ori** talimatı ile implement edilir.

Örnek :

```
.section .text
.global _start


_start:
    movia r2, 5         ; r2 yazmacına 5 değerini yükle
    addi r3, r2, 10     ; r2 yazmacının içeriğine 10 ekle ve sonucu r3
yazmacına kaydet


    ; r3 yazmacının içeriği şimdi 15 olacak (5 + 10 = 15)


    ; Programın sonu, sonsuz döngüye gir
    br _start
```

9. Karşılaştırma Komutları (Comparison Instructions)

Nios II işlemcisindeki **Comparison Instructions** (Karşılaştırma Talimatları), iki kaydın karşılaştırılmasını sağlar. Karşılaştırma sonucuna göre hedef kayda **1 (doğru)** veya **0 (yanlış)** değeri yazılır.

1. Compare Less Than Signed (İşaretli Küçüktür Karşılaştırması)

cmplt rC, rA, rB

- **İşlev:** rA < rB karşılaştırmasını işaretli (signed) sayılar üzerinden yapar.
- **Sonuç:** Eğer rA değeri rB değerinden küçükse, rC'ye **1** yazılır, aksi halde **0** yazılır.

2. Compare Less Than Unsigned (İşaretsiz Küçüktür Karşılaştırması)

cmpltu rC, rA, rB

- **İşlev:** rA < rB karşılaştırmasını işaretsiz (unsigned) sayılar üzerinden yapar.
- **Sonuç:** Eğer rA değeri rB değerinden küçükse, rC'ye **1** yazılır, aksi halde **0** yazılır.

3. Compare Equal (Eşitlik Karşılaştırması)

cmpeq rC, rA, rB

- **İşlev:** rA == rB karşılaştırmasını yapar.
- **Sonuç:** Eğer iki değer eşitse, rC'ye **1** yazılır, aksi halde **0** yazılır.

4. Compare Not Equal (Eşit Değildir Karşılaştırması)

cmpne rC, rA, rB

- **İşlev:** rA != rB karşılaştırmasını yapar.
- **Sonuç:** Eğer iki değer eşit değilse, rC'ye **1** yazılır, aksi halde **0** yazılır.

5. Compare Greater Than or Equal Signed (İşaretli Büyük Eşittir Karşılaştırması)

cmpge rC, rA, rB

- **İşlev:** rA >= rB karşılaştırmasını işaretli (signed) sayılar üzerinden yapar.

- **Sonuç:** Eğer rA, rB'ye büyük veya eşitse, rC'ye **1** yazılır, aksi halde **0** yazılır.

Özet Tablo :

Komut	**Açıklama**	**İfade**
cmplt	İşaretli küçüktür	rA < rB
cmpltu	İşaretsiz küçüktür	rA < rB (u)
cmpeq	Eşittir	rA == rB
cmpne	Eşit değildir	rA != rB
cmpge	İşaretli büyük eşittir	rA >= rB
cmpgeu	İşaretsiz büyük eşittir	rA >= rB (u)
cmpgt	İşaretli büyüktür	rA > rB
cmpgtu	İşaretsiz büyüktür	rA > rB (u)
cmple	İşaretli küçük eşittir	rA <= rB
cmpleu	İşaretsiz küçük eşittir	rA <= rB (u)

10. Dallanma Komutları (Branch Instructions)

Programın yürütme akışı, **Branch (Dallanma)** veya **Jump (Atlama)** talimatları kullanılarak değiştirilebilir. Bu değişiklikler **koşulsuz (unconditionally)** veya **koşullu (conditionally)** olabilir.

1. Jump Talimatı (Koşulsuz Atlama)

jmp rA

Programın yürütme akışını, rA kaydında bulunan adrese **koşulsuz** olarak yönlendirir.

2. Branch Talimatı (Koşulsuz Atlama)

br LABEL

Program yürütmesini, belirtilen LABEL adresine **koşulsuz** olarak dallandırır.

Örnek :

```
_start:
    movia r4, target_address  ; Hedef adresi r4 yazmacına yükle
    jmp r4                    ; Hedef adrese doğrudan sıçra
target_address:
    movia r5, 1               ; Örnek işlem
    br  start                 ; Programın başına dön
```

Koşullu Dallanma (Conditional Branch)

Koşullu dallanma talimatları, iki kaydın içeriğini karşılaştırır ve koşul sağlanırsa dallanmayı gerçekleştirir.

Komut	**Açıklama**	**İfade**
blt	Küçükse dallan	rA < rB
bltu	Küçükse (işaretsiz)	rA < rB (u)
beq	Eşitse	rA == rB
bne	Eşit değilse	rA != rB
bge	Büyük eşitse	rA >= rB
bgeu	Büyük eşitse (işaretsiz)	rA >= rB (u)

Örnek :

```
mov r1, 5 ;

mov r2, 10 ;

blt r1, r2, SMALLER ; r1 değeri, r2 değerinden küçükse, program akışı SMALLER etiketine dallanır.
```

Örnek :

```
mov r1, 5 ;

mov r2, 5 ;

beq r1, r2, EQUAL_LABEL ; r1 ve r2 eşitse, program EQUAL_LABEL adresine dallanır.
```

11. Alt Program (Subroutine)

Nios II işlemci mimarisi, alt programları çağırmak ve geri dönmek için özel talimatlar sağlar. Bu talimatlar, **program akışını alt programlara yönlendirip** işlem tamamlandığında doğru noktaya geri dönmeyi sağlar.

1. call Komutu (J-type) :

call LABEL

Bu talimat, **J-type** formatında çalışır. **26-bit IMMED26** (immediate) değeri içerir.

Alt program çağrılmadan önce, **dönüş adresi (sonraki talimatın adresi) r31 register'ına kaydedilir.**

Adres Hesaplama :

```
Jump address = PC31-28 : IMMED26 : 00

- PC31-28: Program sayacının en yüksek 4 biti.

- IMMED26: Talimatta verilen 26 bitlik anlık değer.

- 00: Talimatların kelime hizalaması gereği en düşük iki bit sıfırdır.
```

Örnek :

```
call FUNCTION ; Dönüş adresi r31'e kaydedilir ve kontrol FUNCTION etiketine gider.
```

2. callr Komutu (R-type) :

callr rA
Bu talimat, R-type formatında çalışır. Alt programın adresi, doğrudan rA kaydında bulunur. Dönüş adresi yine **r31** kaydına kaydedilir. Kontrol, **rA'de belirtilen adrese aktarılır.**

3. ret Komutu (Alt Programdan Dönüş Talimatı) :

ret
Dönüş adresi, **r31** kaydından alınır. Kontrol, r31'deki adrese geri döner.

Örnek :

Bu kod örneğinde, kesme servis rutini, bir kesme meydana geldiğinde interrupt_flag bayrağını manuel olarak 1 yapar. Programcı, kesme bayraklarını manuel olarak ayarlamak ve güncellemek zorundadır. ISR tamamlandığında, ret komutu ile ana programa geri dönülür. Bu yaklaşım, kesme bayraklarının otomatik olarak güncellenmediği ve programcının bu bayrakları kontrol etmesi gerektiği durumları ele almaktadır. Bu nedenle, kesme bayraklarının doğru bir şekilde ayarlandığından ve güncellendiğinden emin olmak programcının sorumluluğundadır.

```
.section .data
# Kesme bayrağı
interrupt_flag: .word 0

.section .text
.global _start
.global interrupt_handler

# Ana program başlangıcı
_start:
    call setup_interrupt  # Kesme işleyicisini kur
    call main_function    # Ana fonksiyonu çağır

main_loop:
    br main_loop          # Ana program döngüsü

# Kesme işleyicisinin kurulumu
setup_interrupt:
    movia r4, interrupt_handler
    wrctl ctl4, r4        # Kesme vektörünü ayarla
    ret                   # Ana programa geri dön

# Ana fonksiyon
main_function:
    # Ana fonksiyon işlemleri
    # Burada herhangi bir işlem yapılabilir
    ret                   # Ana programa geri dön

# Kesme servis rutini (ISR)
interrupt_handler:
    # Kesme bayrağını ayarla
    movia r4, interrupt_flag
    movi r5, 1
    stw r5, 0(r4)         # Bayrağı 1 yap

    # Kesme işleyicisinden geri dön
    ret
```

12. Carry ve Overflow Tespiti (Carry and Overflow Detection)

Nios II işlemcisinde **Add** ve **Subtract** komutları, hem işaretli (signed) hem de işaretsiz (unsigned) sayılarla işlem yapabilir. Ancak, bu komutlar carry (taşıma) ve arithmetic overflow (aritmetik taşma) gibi durumları otomatik olarak algılayıp bayrak (flag) ayarlamazlar. Yani, işlem sonucu taşıma veya taşma meydana geldiğinde, bu durumları algılamak için ek komutlar kullanmak gereklidir.

- **Carry Tespiti**

  Carry, iki sayıyı toplarken, en yüksek bitin (C31) taşarak bir sonraki en yüksek biti etkileyip etkilemediğini belirtir. Nios II işlemcisinde, bu taşıma durumunu tespit etmek için ek bir komut kullanılabilir.

  Carry Tespiti İçin Kullanılan Yöntem:
  Örneğin, bir toplama işleminde taşma olup olmadığını kontrol etmek için şu adımlar takip edilebilir :

  1. **Add** komutunu çalıştırın :

```
add rC, rA, rB  # rA ve rB'yi topla, sonucu rC'ye yaz
```

  2. Taşıma olup olmadığını tespit etmek için **cmpltu** (compare less than unsigned) komutunu kullanarak, eğer **rC**'deki sayı **rA**'dan küçükse, taşıma olmuş demektir :

```
cmpltu rD, rC, rA  # rC < rA kontrol et, taşıma varsa rD'ye 1 yaz
```

  3. Eğer taşıma durumu bir dallanma (branch) tetiklemeli ise, şu şekilde yapılabilir :

```
add rC, rA, rB     # rA ve rB'yi topla

bltu rC, rA, LABEL  # Eğer taşıma varsa, belirtilen etikete dallan
```

  Bu durumda, bltu komutu, **eğer taşıma gerçekleştiyse (yani rC küçükse rA'dan)**, belirli bir etikete dallanır.

- **Overflow Tespiti (Aritmetik Taşma)**

  Aritmetik taşma, iki pozitif sayının toplamının negatif bir sayı oluşturması ya da iki negatif sayının toplamının pozitif bir sayı oluşturması durumudur. Bu tür taşmalar, sayıların işaret bitlerinin incelenerek tespit edilebilir.

  Overflow Tespiti İçin Kullanılan Yöntem :

1. Add komutunu kullanarak toplama işlemi yapılır :

```
add rC, rA, rB  # rA ve rB'yi topla, sonucu rC'ye yaz
```

2. XOR komutları ile, sonucu ve operandları karşılaştırarak işaret bitlerinin farklı olup olmadığına bakılır. Eğer iki pozitif sayı negatif bir sonuç veriyorsa veya iki negatif sayı pozitif bir sonuç veriyorsa, taşma meydana gelmiş demektir:

```
xor rD, rC, rA  # rC ile rA'yı karşılaştır (işaret bitlerini kontrol et)
xor rE, rC, rB  # rC ile rB'yi karşılaştır (işaret bitlerini kontrol et)
and rD, rD, rE  # İşaret bitlerinin uyuşup uyuşmadığını kontrol et
```

3. Son olarak, overflow meydana gelip gelmediği, rD'deki değere göre kontrol edilebilir. Eğer overflow varsa, rD negatif bir değer olacaktır ve buna göre dallanma yapılabilir:

```
blt rD, r0, LABEL  # Eğer overflow olduysa, belirtilen etikete dallan
```

Burada, **blt** komutu, rD'nin sıfırdan küçük olup olmadığını kontrol eder. Eğer küçükse, overflow durumu meydana gelmiştir ve belirtilen etikete (LABEL) dallanılır.

13. Exception Processing (İstisna İşleme)

Nios II işlemcisinde bir istisna (**exception**), program yürütme akışındaki normal durumu kesintiye uğratan ve işlemcinin özel bir işleyiciye geçmesini gerektiren durumlar için kullanılan bir mekanizmadır. Bir istisna; yazılım tuzağı (**trap**), donanım kesmesi (**hardware interrupt**) veya uygulanmamış talimat (**unimplemented instruction**) gibi nedenlerle tetiklenebilir. Bir istisna meydana geldiğinde Nios II işlemcisi belirli adımları takip eder. İlk olarak, işlemcinin mevcut durum bilgisi, **status register (ctl0)** içeriği **estatus register (ctl1)** içine kopyalanarak saklanır. Ardından, işlemci **Supervisor Modu**na geçer ve **PIE (Processor Interrupt Enable)** biti sıfırlanarak ek harici kesmeler devre dışı bırakılır. İstisnayı tetikleyen talimattan sonraki talimatın adresi **ea (Exception Address) (r29)** kaydına yazılır ve işlemci kontrolü, **istisna işleyiciye (exception handler)** devreder. İstisna işleyicisi, sorunun türünü belirleyerek uygun hizmet rutinini çağırır ve gerekli müdahaleyi gerçekleştirir. İstisna işleyicisinin adresi, SOPC Builder kullanılarak sistem oluşturulurken

belirlenir ve çalışma zamanında yazılım tarafından değiştirilemez. Varsayılan olarak bu adres, ana belleğin başlangıcından itibaren **0x00000020** olarak tanımlıdır.

- **Yazılım Tuzağı (Software Trap)**, trap talimatı ile tetiklenir. Bu talimat çalıştırıldığında, işlemcinin mevcut komut adresi **ea (r29)** kaydına kaydedilir ve işlemci kontrolü istisna işleyicisine devreder. İstisna işleyicisi, sorunu çözdükten sonra eret talimatı ile eski çalışma durumuna geri döner. Bu süreçte işlemcinin önceki durumu **estatus** kaydından geri yüklenir. Yazılım tuzakları genellikle işletim sistemi çağrıları veya hata ayıklama süreçleri için kullanılır.

- **Donanım Kesmesi (Hardware Interrupt)**, harici kaynaklar veya çevre birimleri tarafından tetiklenir. Nios II işlemcisi, 32 adet kesme girişine (**irq0-irq31**) sahiptir. Bir kesmenin gerçekleşebilmesi için **PIE biti etkinleştirilmiş**, ilgili kesme hattı (**irqk**) aktif hale getirilmiş ve ilgili kesme etkinleştirme biti (**ctl3k**) ayarlanmış olmalıdır. Bekleyen kesmeler, **ipending (ctl4)** kaydı üzerinden kontrol edilir ve en yüksek öncelikli kesme işlenir. Kesme hizmet rutini tamamlandıktan sonra kontrol eret talimatı ile ana programa geri döner. Ancak kesmeler, kesme anındaki talimatın tamamlanmasını beklemediği için **ea** kaydındaki değer, doğru noktaya geri dönmek için **4 bayt azaltılmalıdır**.

- **Uygulanmamış Talimat (Unimplemented Instruction)**, işlemcinin donanım düzeyinde desteklemediği bir talimatla karşılaşıldığında ortaya çıkar. Örneğin, mul veya div gibi talimatlar bazı işlemci varyantlarında donanım desteğiyle çalışmaz ve bu durumda işlemci bir istisna üretir. Bu tür talimatların işlevselliği, yazılım tarafından sağlanan bir rutin aracılığıyla taklit edilir. Bir istisna meydana geldiğinde işlemci, uygun hizmet rutinini çağırmak için istisna türünü belirler. İlk olarak, **ipending (ctl4)** kaydı okunarak donanım kesmesi olup olmadığı kontrol edilir. Eğer donanım kesmesi yoksa, işlemci **ea** kaydındaki adrese bakarak talimatın trap olup olmadığını kontrol eder. Eğer bu iki koşul da sağlanmıyorsa, istisnanın nedeni uygulanmamış bir talimat olarak kabul edilir ve yazılım emülasyonu başlatılır. Bu mekanizma, Nios II işlemcisinin beklenmedik durumlara doğru ve verimli bir şekilde tepki vermesini sağlar.

14. Nios II Assembly Dilinde Assembler Yönergeleri (Assembler Directives)

Assembler yönergeleri, derleyiciye (assembler) kaynak kodunun nasıl işlenmesi gerektiğini belirten talimatlardır. Bu yönergeler doğrudan işlemci tarafından çalıştırılmaz, yalnızca derleme sürecini yönlendirmek için kullanılırlar. GNU Assembler (GAS) tarafından desteklenen bu yönergeler, Nios II derleyicisinde de kullanılabilir.

- **.byte expressions**
  Virgülle ayrılmış ifadeleri bayt (8-bit) olarak belleğe yazar. İfadeler sayısal değerler, sabitler veya etiketlere yapılan işlemler olabilir.
  Örnek :

```
.byte 8, 5 + LABEL, K - 6
```

- **.end**
  Kaynak dosyanın sonunu belirtir. Bu yönergede sonra gelen her şey derleyici tarafından yok sayılır.
  Örnek :

```
.end
```

- **.equ symbol, expression**
  Bir sembolün değerini bir ifadeye eşitler. Bu, makrolar ve sabit değerler tanımlamak için kullanılır.
  Örnek :

```
.equ BUFFER_SIZE, 256
```

- **.global symbol**
  Bir sembolü dışarıdan erişilebilir hale getirir. Diğer dosyalardan bu sembole erişim sağlanabilir.
  Örnek :

```
.global main; main sembolü dışarıdan erişilebilir hale gelir.
```

- **.include "file"**
  Başka bir dosyanın içeriğini mevcut kaynak dosyaya dahil eder. Genellikle tanımlar veya makrolar içeren başlık dosyaları için kullanılır.

Örnek :

```
.include "config.s"
```

- **.org new-lc**
  Konum sayacını (**Location Counter**) belirtilen adrese ilerletir. Sayacın geriye hareket etmesine izin verilmez. Genellikle belirli bir bellek adresinden veri veya kod başlatmak için kullanılır.
  Örnek :

```
.org 0x1000 ; Kod veya veri, bellekte 0x1000 adresinden başlar.
```

15. Örnek Uygulama

Alt sayfada yer alan NIOS II işlemcisi için yazılmış bu assembly programı, temel bir sayaç uygulaması gerçekleştirmektedir. Program, LED'leri kontrol etmek için belirli bir I/O adresini (örnekte 0x810) kullanarak, sıfırdan 255'e kadar sayan ve bu değerleri LED'lerde gösteren bir sistem oluşturur. Programın yapısı, .text bölümüyle başlayıp global _start etiketi ile işlemciye başlangıç noktasını belirtir. Ana program, LED'lerin kontrolü için gerekli I/O adresini r2 registerına yükleyerek ve sayaç değerini tutacak r3 registerını sıfırlayarak başlar.

Programın çalışma mantığı, sürekli tekrar eden bir döngü içerisinde sayaç değerini artırma ve bu değeri LED'lere yansıtma prensibine dayanır. Her döngüde, mevcut sayaç değeri LED'lere yazıldıktan sonra, görsel olarak değişimin algılanabilmesi için bir gecikme rutini uygulanır. Bu gecikme rutini, r4 registerında tutulan 1 milyon değerinin sıfıra kadar azaltılması işlemini gerçekleştirir. 50 MHz'lik bir NIOS II işlemcisinde bu gecikme yaklaşık 60 milisaniyelik (0.06 saniye) bir süreye karşılık gelir. Gecikme süresi, işlemcinin clock hızına bağlı olarak değişiklik gösterebilir ve daha uzun süreli gecikmeler için değer artırılabilir. Örneğin, bir saniyelik gecikme için yaklaşık 16,666,667 değeri kullanılması gerekir.

Program içerisinde kullanılan NIOS II komutları arasında movia (adres yükleme), movi (değer yükleme), stwio (I/O portuna yazma), subi (çıkarma), addi (toplama), bne (eşit değilse dallanma) ve br (koşulsuz dallanma) bulunmaktadır. Register kullanımında r2 I/O adresini, r3 sayaç değerini, r4 gecikme sayacını ve r5 üst limit değerini tutarken, r0 NIOS II mimarisinde standart olarak sıfır değerini tutan registerdır. Sayaç değeri 256'ya ulaştığında (8-bit sınırı), program otomatik olarak sayacı sıfırlar ve döngüyü yeniden başlatır.

Bu programı test etmek için https://cpulator.01xz.net/?sys=nios adresindeki çevrimiçi NIOS II simülatörü kullanılabilir. Bu platform, gerçek donanım olmadan NIOS II assembly kodlarını çalıştırma ve davranışlarını analiz etme imkanı sağlar. Simülatör üzerinde kodun çalışması gözlemlenebilir, register değerleri takip edilebilir.

```
.section .text       # Kodun text (program) bölümünün başlangıcını belirtir
.global _start       # _start etiketini global yapar, böylece linker bu noktadan başlar
_start:              # Programın başlangıç noktası
    # LED'leri kontrol etmek için I/O adresi ayarlanıyor
    movia r2, 0x810 # r2 registerına LED'lerin bağlı olduğu I/O adresini yükler
                    # 0x810 örnek bir adrestir, gerçek donanıma göre değiştirilmeli
    movi r3, 0      # r3 registerı sayaç olarak kullanılacak
                    # Başlangıç değeri olarak 0 yükleniyor
counter_loop:       # Ana döngü etiketi
    stwio r3, 0(r2) # Store Word I/O komutu
                    # r3'teki sayaç değerini, r2'de tutulan I/O adresine yazar
                    # Böylece LED'ler sayaç değerine göre yanar
    # Gecikme (Delay) rutini başlangıcı
    movia r4, 1000000  # r4'e 1 milyon değerini yükler
                       # Bu değer gecikme süresini belirler
                       # Sistem clock hızına göre ayarlanmalıdır
delay:              # Gecikme döngüsü etiketi
    subi r4, r4, 1  # r4'ten 1 çıkarır (subtract immediate)
    bne r4, r0, delay # Branch if Not Equal komutu
                      # Eğer r4 sıfır değilse, delay etiketine geri döner
                      # Bu şekilde gecikme sağlanır
    # Sayaç değerini artırma
    addi r3, r3, 1  # r3'e 1 ekler (add immediate)
                    # Böylece sayaç bir artırılmış olur
    # Sayaç üst limit kontrolü
    movi r5, 256    # r5'e 256 değerini yükler
                    # 8-bit LED dizisi için üst limit
    bne r3, r5, counter_loop  # Eğer sayaç 256'ya ulaşmadıysa
                              # counter_loop'a geri dön
    movi r3, 0     # Eğer sayaç 256'ya ulaştıysa
                   # r3'ü (sayacı) sıfırla
    br counter_loop # Branch (koşulsuz dallanma) komutu
                    # Ana döngüye geri dön
```

16. Görüş Ve Öneriler

Nios II kullanım kılavuzunu incelediğimde, FPGA tabanlı tasarımlar için oldukça güçlü bir işlemci seçeneği sunduğunu fark ettim. Kılavuzun kapsamı genel olarak iyi olmakla birlikte, bazı alanlarda daha fazla ayrıntı ve açıklama eklenmesi, özellikle FPGA ile yeni tanışan kişiler için faydalı olabilir. Nios II'nin özelleştirilebilirliği gerçekten dikkat çekici, ancak bazı özelliklerin nasıl daha derinlemesine kullanılabileceği konusunda daha fazla örneğe yer verilmesi gerektiğini düşünüyorum. Özellikle işlemcinin farklı konfigürasyonlarını seçerken ya da özelleştirirken yaşanabilecek zorlukları aşmak adına, daha fazla pratik örnek ve adım adım rehber sağlanabilir.

Bir diğer önemli nokta, yazılım geliştirme süreciyle ilgili olan kısmın biraz daha derinlemesine ele alınması gerektiği. Kılavuzda, Nios II ile yazılım geliştirmeye başlamak için temel adımlar gayet açık, ancak yazılımın FPGA üzerinde çalıştırılması ve debug edilmesi konusunda biraz daha fazla bilgi verilebilirdi.

Ayrıca, kılavuzun özellikle FPGA üzerinde çalışmaya yeni başlamış kişiler için daha anlaşılır bir dilde yazılabileceğini düşünüyorum. Teknik terimler yer yer oldukça yoğun ve ilk defa FPGA ile çalışacak birinin kafasında karmaşıklık yaratabilir. Bu terimlerin ne anlama geldiği, ne amaçla kullanıldıkları gibi açıklamaların daha belirgin şekilde verilmesi, herkesin metni rahatlıkla anlayabilmesini sağlar.

Bunun yanı sıra, kılavuzda FPGA ile Nios II'nin entegrasyonu konusunda örnek bulunması da çok yararlı olabilir. Diğer platformlarla ve cihazlarla olan entegrasyon süreci bazen kafa karıştırıcı olabilir, özellikle çok farklı cihazlarla çalışırken. Bu konuda daha fazla bilgi verilmesi, kullanıcıların Nios II'yi daha geniş bir sistemde kullanırken karşılaşabilecekleri zorlukları aşmalarına yardımcı olacaktır.

Son olarak, kılavuzun görsel desteği de oldukça faydalı. Ancak, görsellerin daha fazla yer aldığı, her adımın görsel olarak desteklendiği bir versiyon, kullanıcı deneyimini çok daha verimli hale getirebilir. Özellikle devrelerin nasıl kurulduğunu, işlemcinin nasıl konfigüre edildiğini veya yazılımın nasıl yüklendiğini görselleştiren daha fazla şema ve ekran görüntüsü, her seviyedeki kullanıcı için oldukça faydalı olacaktır.

Genel olarak, Nios II kullanım kılavuzu oldukça kapsamlı ve güçlü bir içerik sunuyor. Yine de, daha fazla açıklama, örnek ve görsel desteği ile, kullanıcıların daha kolay öğrenmesini sağlamak mümkün olacaktır.

17. Kaynakça

- Introduction to the Altera Nios II Soft Processor [Kaynak Bağlantı: https://people.ece.cornell.edu/land/courses/ece5760/DE2/tut_nios2_introduction.pdf]

- Nios II Processor Reference Handbook [Kaynak Bağlantı: https://www-ug.eecg.toronto.edu/msl/manuals/n2cpu_nii5v1.pdf]

- Nios® II 32-bit RISC Processor [Kaynak Bağlantı: https://valhalla.altium.com/Learning-Guides/Legacy/CR0164%20Nios%20II%2032-bit%20RISC%20Processor.PDF]

- Nios® II Processor Reference Guide [Kaynak Bağlantı: https://cdrdv2-public.intel.com/666887/n2cpu-nii5v1gen2-683836-666887.pdf]

Bilgilendirme :
Bu rapor, Nios II işlemcisi hakkında Türkçe bir kaynak oluşturmak amacıyla GitHub üzerinden herkese açık olarak paylaşılmıştır.

https://github.com/alikank/nios-II-kilavuz