TOBB Ekonomi ve Teknoloji Üniversitesi
Bilgisayar Mühendisliği Bölümü
BİL 361 – Bilgisayar Mimarisi ve Organizasyonu

07 Şubat 2025
2024 – 2025 Öğretim Yılı
Bahar Dönemi
Ödev 1

[140 puan] Çok Vuruşluk RISC-V İşlemci Tasarımı

Bu ödevde RISC-V RV32I buyruk kümesi mimarisindeki bazı buyruklarını (a şıkkı) ve üç tane özel buyruk (b şıkkı) yürütebilen çok vuruşluk 32-bit işlemci tasarlayacaksınız.

İşlemcinizin desteklemesi gereken buyruklar aşağıdaki tabloda belirtilmiştir. İşlemcide olacak buyrukların ayrıntılı açıklamalarına ve bu tablonun tüm RISCV buyruklarını içeren haline unpriv-isa-asciidoc.pdf bağlantısından ulaşabilirsiniz.

RV32I Base Instruction Set

imm[31:12]				rd	0110111	LUI
imm[31:12]				rd	0010111	AUIPC
imm[20|10:1|11|19:12]				rd	1101111	JAL
imm[11:0]		rs1	000	rd	1100111	JALR
imm[12|10:5]	rs2	rs1	000	imm[4:1|11]	1100011	BEQ
imm[11:0]		rs1	010	rd	0000011	LW
imm[11:5]	rs2	rs1	010	imm[4:0]	0100011	SW
imm[11:0]		rs1	000	rd	0010011	ADDI
0000000	rs2	rs1	000	rd	0110011	ADD
0100000	rs2	rs1	000	rd	0110011	SUB
0000000	rs2	rs1	110	rd	0110011	OR
0000000	rs2	rs1	111	rd	0110011	AND
0000000	rs2	rs1	100	rd	0110011	XOR

- RISC-V Buyruk Kümesi Mimarisi tüm buyrukların verimli ve etkin çalışması üzerine tasarlanmıştır. Fakat problemi kolaylaştırmak adına size 13 adet buyruk ile çalışabilecek bir işlemci tasarlama görevi verilmiştir. Buyruk mimarisinde belirtilen ve bu 13 buyruk dışındaki şartlardan dolayı oluşan tüm kısıtlamaları ve kural dışı durumları yok sayabilirsiniz.

- Yapacağınız işlemci belirtilen tüm buyruklar için çok çevrimde doğru işlemleri tamamlamalıdır. Bu aşamalar sırasıyla "**Getir**", "**ÇözYazmaçOku**" ve "**YürütGeriYaz**" olacaktır ve. "**Getir**" ve "**ÇözYazmaçOku**" yalnızca **bir** çevrim sürmelidir, "**YürütGeriYaz**" aşaması buyruğa bağlı şekilde **bir veya daha çok çevrimde gerçekleşebilir** (örn., b şıkkındaki Kirby Sort buyruğu çok çevrimli gerçeklenmelidir.). Aşamaların işlevleri aşağıda detaylıca açıklanmıştır. Kafanızı karıştıran durumlarda piazzada ödev gönderisini kullanmaya çekinmeyin.
    - **Getir:** İşlenmesi gereken buyruğun program sayacı belleğe gönderilir ve gelen buyruk bir sonraki aşama için kayıt edilir. Getir aşaması istek yapıldıktan sonra saatin yükselen kenarında program sayacını günceller.
    - **ÇözYazmaçOku:** Getirilen buyruk mikroişlemlere ayrılır (örn, AMB için yapılacak işlemin kayıt edilmesi) ve ilgili yazmaçlar okunarak bir sonraki aşama için kayıt edilir. Dallanma işlemlerini bu aşamada **sonuçlandırmayın**.
    - **YürütGeriYaz:** Çözülen mikroişlemlere göre buyruk yürütülür ve durum (yazmaçlar, ana bellek) güncellenir. Dallanma yapılıyorsa YürütGeriYaz aşaması bir sonraki buyruk için program sayacını düzgün şekilde günceller.

- İşlemci ile ilgili genel kısıtlar aşağıdaki gibidir, bu tanımlara uyduğunuzdan emin olmak için kaynaklar altındaki taslak tasarım dosyalarını kullanabilirsiniz.
  - İşlemci ve bellekte bayt adresleme kullanılır.
  - 32 adet 32 bitlik yazmaç bulunur ve değişkenin adı "**yazmac_obegi"**dir.
  - **simdiki_asama_r** yazmacı her zaman o çevrimde yürütülen aşamayı gösterir.
  - **ps_r** yazmacı her zaman işlenecek sıradaki buyruğun adresini gösterir (dallanmalarla değişebilir).
  - **ilerle_cmb** çıktısı, işlemcinin aşamalarının ilerleyip ilerlemediğini gösterir. Örn. Getir aşamasındayken ilerle_cmb çıktısı mantık-1 vermelidir çünkü Getir aşaması tek çevrim sürer ve bir sonraki aşamaya ilerler. YürütGeriYaz aşamasında ise sadece aşamanın biteceği çevrimde ilerle_cmb çıktısı mantık-1 vermelidir.

  Örnek aşama akışı:

çevrim	0	1	2	3	4	5	6	7	8
simdiki_asama_r	Getir	ÇözYazmaçOku	Yürüt GeriYaz	Yürüt GeriYaz	Yürüt GeriYaz	Getir	ÇözYazmaçOku	Yürüt GeriYaz	…
ilerle_cmb	mantık-1	mantık-1	mantık-0	mantık-0	mantık-1	mantık-1	mantık-1	mantık-0	…

- 2048 satırlı 32 bitlik elemanlardan oluşan ana bellek, işlemciye "**bellek_adres**" çıkışında bulunan adresteki veriyi verecektir. Ana bellekteki ilk elemanın adresi "0x8000_0000" olup programın ilk buyruğu her zaman bu adreste yer alacaktır.

Dikkat: Çok vuruşluk bir işlemci tasarladığınız için **bellek_adres** çıkışında **Getir** aşaması için program sayacı olup buyruk getirilecek, **ÇözYazmaçOku** aşamasında ise buyruk çözülüp gerekli yazmaç değerlerini okunacak, **YürütGeriYaz** aşamasında ise çözülen buyruk yürütülüp gerekli yazmaç ve bellek adresleri düzenlenecek, ve bir sonraki buyruğun adresi belirlenecektir.

Not: Önemli bir nokta değil ancak dikkatinizi çektiği üzere programlar istenirse kendi buyruklarının üzerine veri yazarak programı bozabilirler / değiştirebilirler. İşletim sistemleri normal yazdığınız programlarda bunu zorlaştırsa da bu durum olağandır ve engellenmesine gerek yok.

[70 Puan] a) Yukarıda ayrıntıları verilen çok vuruşluk işlemciyi verilog dilinde "**islemci.v**" ile tek bir modül olarak tasarlayınız. "**islemci.v**" için taslak kodunu piazza kaynaklar kısmında bulabilirsiniz. Bu dosyaya giriş / çıkışlar, yazmac_obegi, simdiki_asama_r ve ps_r değişkenlerinin **isimleri** dışında istediğiniz değişikliği yapabilirsiniz (sadece isimler önemli, atamaları ve genel kodu değiştirmenizde bir sakınca yok).
işlemci.v giriş çıkışları sırasıyla aşağıdaki gibidir.

- **clk** : 1 bitlik saat girişi
- **rst** : 1 bitlik işlemcinizi başlangıç durumuna geri çeviren giriş
- **bellek_adres** : 32 bitlik belleğe giden adres çıkışı
- **bellek_oku_veri** : 32 bitlik bellekten okunan veri girişi
- **bellek_yaz_veri** : 32 bitlik belleğe yazılacak veri çıkışı
- **bellek_yaz** : 1 bitlik belleğe yazma yapılacağını gösteren çıkış
- **ilerle_cmb** : 1 bitlik çıkış

- Bellekte yazma işlemleri **sıralı mantık**, okuma işlemleri ise **kombinasyonel** mantıkla çalışmaktadır. Yani ilgili çıkışlar atandıktan sonra **yazma işlemleri saatin yükselen kenarında** gerçekleşirken **okuma işlemlerinde saatin yükselen kenarı beklenmeden bellek_oku_veri** girişinde olacaktır. Kaynaklar altında "**anabellek.v**" dosyasını inceleyip testlerinizde kullanabilirsiniz.

Önemli Not: Bellekten sadece 4 bayt hizalı okumalar yapılacağını varsayabilirsiniz. Yani 0x8000_0002 gibi bir adres üzerinde **okuma** veya **yazma** asla gerçekleştirilmeyecektir. Ek olarak 0x8000_0000 adresi bellekteki ilk 4 baytın başlangıcını, 0x8000_0004 ise ikinci 4 baytın başlangıcını adresleyecektir.

Önemli Not: Buyruklarda kullanılan anlık değerleri RISC-V spesifikasyonuna uygun bir şekilde **işaretle genişletmeye** (*-ing.,* sign-extension) dikkat edin.

Not: Buyrukların ne yaptığını açıklayan başka bir siteye RISC-V Instruction Set Specifications bağlantısından ulaşabilirsiniz.

Not: RISC-V Buyruklarını kodlayan ve kodunu çözen bir siteye de rvcodecjs bağlantısından ulaşabilirsiniz.

[70 Puan] b) a şıkkındaki işlemciyi aşağıdaki üç buyruğu yürütecek şekilde değiştirin. Yeni işlemciyi verilog dilinde "**islemci_yusf.v**" adında tek bir modül olarak tasarlayınız. "**islemci_yusf.v**" için "**islemci.v**" taslak kodunu kullanabilirsiniz. Bu dosyaya giriş / çıkışlar, yazmac_obegi, simdiki_asama_r ve ps_r değişkenlerinin **isimleri** dışında istediğiniz değişikliği yapabilirsiniz (sadece isimler önemli, atamaları ve genel kodu değiştirmenizde bir sakınca yok).

Not: Yeni buyrukların **"YürütGeriYaz"** aşamalarının çok çevrimli olması zorunludur. **"YürütGeriYaz"** aşaması tek çevrimde sonuçlanan buyruk gerçeklemelerine **puan verilmeyecektir**.

Not: Normalde yazmaç okumaları **"ÇözYazmaçOku"** aşamasında yapılmaktadır. Kirby Sort buyruğunu eklerken **"YürütGeriYaz"** aşamasında yazmaç okumanız gerekebilir ve buna izin verilmektedir.

- LIS (Load Increment Store)

imm[11:0]	rs1	010	incr	1110011

LIS buyruğu **rs1 + imm** bellek adresindeki değeri okur, bu değer ile **incr** arasında *işaretli* toplama yapar ve yeniden rs1 + imm bellek adresine yazar. Buyruk bellekten okuduğu değeri herhangi bir yazmaca **yazmaz**.

Not: Bellek adresleri hesaplanırken işaretli toplama yapın. Anlık değerleri genişletirken işaretli genişletme kullanın.

Örn. imm = 12, rs1 = 5, incr = 15 (rs1 değerinin 0x8000_0100 olduğunu varsayın.)

İşlem öncesinde 0x8000_0100 adresi çevresindeki 16 bayt (4 baytlar gruplanmıştır. En sağdaki bayt en anlamsız baytı göstermektedir):

Bellek Adresi	Tutulan Değer
0x8000_0100	**0x00_00_00_09**
0x8000_0104	**0x00_00_00_0A**
0x8000_0108	**0x00_00_00_0B**
0x8000_010C	**0x00_00_00_0C**

Buyruk çalıştırıldıktan sonra gözlemlenmesi gereken mimari durumu (*-ing.,* architectural state) sonraki sayfada gösterilmiştir.

İşlem sonrasında 0x8000_0100 adresi çevresindeki 16 bayt (4 baytlar gruplanmıştır. En sağdaki bayt en anlamsız baytı göstermektedir):

Bellek Adresi	Tutulan Değer
0x8000_0100	**0x00_00_00_09**
0x8000_0104	**0x00_00_00_0A**
0x8000_0108	**0x00_00_00_0B**
0x8000_010C	**0x00_00_00_1B**

LLM (Load Load Multiply)

imm[11:0]	rs1	011	rd	1110011

LLM buyruğu **rs1 + imm** ve **rs1 + imm + 4** bellek adreslerinden 4 bayt hizalı okuma yapar, bu iki değer arasında *işaretli* çarpma yapar ve sonucu rd yazmacına yazar.

Not: Bellek adresleri hesaplanırken işaretli toplama yapın. Anlık değerleri genişletirken işaretli genişletme kullanın.

Örn. imm = 4, rs1 = 5, rd = 15 (rs1 değerinin 0x8000_0100 olduğunu varsayın.)

0x8000_0100 adresi çevresindeki 16 bayt (4 baytlar gruplanmıştır. En sağdaki bayt en anlamsız baytı göstermektedir):

Bellek Adresi	Tutulan Değer
0x8000_0100	0x00_00_00_09
0x8000_0104	0x00_00_00_0A
0x8000_0108	0x00_00_00_0B
0x8000_010C	0x00_00_00_0C

İşlem sonrasında 16. yazmaca (x15) 110 değeri yazılır.

- KS (Kirby Sort)

0000000	length	rs1	001	rd	1110011

KS buyruğu rs1'den başlayarak length (5 bit) değeri kadar yazmacı sıralar ve rd yazmacından itibaren yazar. Buyruk rs1'den başlayarak length değeri kadar yazmacın değerlerini **değiştirmez**, sıralanmış değerleri rd yazmacından itibaren kaydeder. Sıralama yaparken küçükten büyüğe sıralanır ve bir önceki yazmaçlardan küçük olan yazmaçlar **kaydedilmez**. rd'nin buyruğun işlevini bozmayacak şekilde seçildiğini varsayabilirsiniz (rd bu şekilde seçildiyse bile bu durumu düzeltmeniz gerekmiyor).

Buyruk çalıştırıldıktan sonra gözlemlenmesi gereken mimari durumu sonraki sayfada gösterilmiştir.

Örn. rs1 = 5, length = 6, rd = 11

Yazmaç ID	Buyruk yürütülmeden önce ilgili yazmaçların değerleri	Buyruk yürütüldükten sonra ilgili yazmaçların değerleri	
5	7	7	İlk yazmaç her zaman kaydedilir (yazmac_11).
6	9	9	9, 7'den büyük olduğu için yazmaç kaydedilir (yazmac_12).
7	8	8	8, 9'dan küçük olduğu için yazmaç kaydedilmez.
8	15	15	15, 9'dan büyük olduğu için yazmaç kaydedilir (yazmac_13).
9	13	15	15, 15'ten büyük olmadığı için yazmaç kaydedilmez.
10	17	17	17, 15'ten büyük olduğu için yazmaç kaydedilir (yazmac_14).
11	0	7	
12	0	9	
13	0	15	
14	0	17	
15	0	0	

Ödev Gönderimi ve Formatı

Ödevinizde işlemci tasarımı için yazdığınız Verilog dosyaları ve **islemci** tasarımlarınızın sentez çıktısı bulunmalıdır. Ödevinizi uzak platformuna **sıkıştırmadan** yükleyeceksiniz.

Gönderdiğiniz tüm tasarımların **sentezlenebilir** verilog standartlarında yazılması beklenmektedir. Tasarımlarınızın keyfi bir FPGA kartı için sentezlenebilir olduğunu kontrol edin. Modülleriniz simülasyon üzerinden kontrol edileceğinden implementasyon ve bitstream aşamaları **gerçekleştirilmeyecek**. Bu aşamalar için "**constraints**" dosyaları oluşturmakla **vakit kaybetmeyin**.

Sentez çıktılarını almak için bir çok farklı yol var ancak daha önce yapmadıysanız aşağıdaki adımları izleyebilirsiniz:

-Proje ağacından ilgili modüle (islemci.v) sağ tıklayıp "**Set as Top**"ı seçin. Eğer zaten ilgili modül kalın fontla yazılıysa (zaten hedef olarak seçiliyse) bu adımı atlayabilirsiniz.
-Vivado'da sol kısımdaki menüden "**SYNTHESIS"** altında bulunan "**Run Synthesis**" seçeneğine tıklayın.
-Senteziniz tamamlandığında sentez raporunuzu (vds dosyasını) kayıt edin ve gönderiminize ekleyin. Raporunuz sentez bittikten sonra aşağıdaki yolda oluşacaktır.

"**{PROJE_KLASORU}/{PROJE_ISMI}.runs/synth_1/{MODUL_ISMI}.vds**"

Yüklenecek dosyalar: islemci.v, islemci_yusf.v, islemci.vds, islemci_yusf.vds

Son Teslim Tarihi: 7 Mart 2025, 23:59