T.C.
BURSA ULUDAĞ ÜNİVERSİTESİ
MÜHENDİSLİK FAKÜLTESİ
BİLGİSAYAR MÜHENDİSLİĞİ

Murtaza CİCİOĞLU
(Dersin Öğretim Görevlisi)

BMB2014
PYTHON PROGRAMLAMAYA GİRİŞ DERSİ
PROJE ÖDEV RAPORU

1. Grup

Damla SOYDAN	032190059
Ferit Yiğit BALABAN	032190002
İrem İÇÖZ	032190009
Sabir SÜLEYMANLI	032190134
Zeynep KILINÇER	032190074

İÇİNDEKİLER

ÖZET

RISC-V ISA'inin 32I Genişletmesi'nden esinlenilerek RISC (Reduced Instruction Set Computer) bir Komut Küme Mimarisi (Instruction Set Architecture, ISA) tasarlanmış, 32 yazmaçlı ve 5 bayraklı bir mikroişlemci üzerine kurgulanmıştır. Tasarlanan komut küme mimarisi (buradan sonra KKM olarak kısaltılacaktır) Assembly dilinin x86 ve RISCV tanımındaki dialektleri göz önünde bulundurularak, özel tasarladığımız KKM için bir transpiler ve parser ile komutları çalıştıran simülasyon ortamı var edilmiştir. "RISC-Mini" olarak adlandırdığımız yeni dialekt, temelde 21 komuttan oluşmaktadır.
Simülasyon ortamı hem grafik hem de konsol olarak iki farklı formda var edilmiştir. Grafik arayüz tasarımı için Figma, kodlanması için Python'ın Tkinter kütüphanesi kullanılmıştır.

RISC-Mini: Sanallaştırılmış Mikroişlemci Yorumlama Ortamı

1. GİRİŞ

1.1. Proje Önerisinin Önemi

Günümüzde mikroişlemci tasarımı öğrenmeye başlamanın en doğru yolu temellerde yer alan tarihi işlemcilerin ve yonga setlerinin nasıl icat edildiğini incelemek ile mümkün olacaktır. Bu noktada yazdığımız simülasyon araçları, gerekli olan eğitim ve deney ortamlarını, fiziksel olarak o yongaya sahip olmayanların da kullanmasını kolaylaştırarak fırsat eşitliği sağlamaktadır. Bir mikroişlemcinin iç işleyişini dünyayı değiştirmiş gerçek bir yongayı deneyimleyerek öğrenmek, bu aracı kullanan öğrencilerin ilerleyen çalışmaları ve araştırmaları için anektodal bilgi edinmelerini sağlayacaktır.

1.2. Hedefler

Gerçek zamanlı bir mikroişlemci simülasyonu sunmak, mikroişlemciye özgü Assembly dili ile alınan komut girişlerini anında makine kodu olarak görüntülemek ve sanal bir bellek üzerinde değişiklikler yaparak gerçek bir bilgisayarın donanımsal arayüzünü kolayca erişilebilir ve deney ortamlarında kullanılabilir hale getirmek.

2. YAZILIMIN DETAYLARI

2.1. Mikroişlemcinin İç Yapısı

2.1.1. Yazmaçlar

İşlemci x0'dan başlayarak x31'e kadar sıralanmış, 32 adet, 32-bit genişliğinde işaretli tam sayı tutan genel amaçlı yazmaca sahiptir.

2.1.2. Bayraklar

Bayrak Kısaltması	Açıklaması
C	Elde (carry)
Z	Sıfır (zero)
S	İşaret (sign)
V	Taşma (overflow)
XLEN	Tam sayı formunda 32 değerini tutar

Yazmaçların yanında C, Z, S ve V ile adreslenebilen bayrak yazmaçları yer almaktadır. Bayraklar da 32-bit genişliğindedir fakat yalnızca 0h ve 1h değerlerini saklamak için kullanılmalıdır.

2.2. RISC-Mini Assembly Dialekti Sentaksı ve KKM

2.2.1. Aritmetik Komutlar

Mnemonik	Sentaks	Açıklama
add	`add rd, rs1, rs2`	rd ← (rs1 + rs2)
inv	`inv rd`	rd ← (-1 * rd)

sub	`sub rd, rs1, rs2`	rd ← (rs1 - rs2)
slt	`slt rd, rs1, rs2`	rd ← (rs1 < rs2 ? rs1 : rs2)
nop	`nop`	x0 ← x0

2.2.2. Mantıksal Komutlar

Mnemonik	Sentaks	Açıklama
and	`and rd, rs1, rs2`	rd ← (rs1 & rs2)
or	`or rd, rs1, rs2`	rd ← (rs1 | rs2)
xor	`xor, rd, rs1, rs2`	rd ← (rs1 ^ rs2)
shl	`shl rd, rs1, rs2`	rd ← (rs1 << rs2)
shr	`shr rd, rs1, rs2`	rd ← (rs1 >> rs2)

2.2.3. Dallanma Komutları

Mnemonik	Sentaks	Açıklama
jmp	`jmp section`	x30 yazmacına program sayacını keydederek sectiona atlar
beq	`beq rs1, rs2, section`	rs1==rs2 ? jmp section : nop
bne	`bne rs1, rs2, section`	rs1!=rs2 ? jmp section : nop
bge	`bge rs1, rs2, section`	rs1>=rs2 ? jmp section : nop
ble	`ble rs1, rs2, section`	rs1<=rs2 ? jmp section : nop

2.2.4. Bellek Komutları

Mnemonik	Sentaks	Açıklama
lfm	`lfm rd, [hex_value]h`	Belleğin hex_value adresinde yer alan değerini rd yazmacına yükler
stm	`stm rd, [hex_value]h`	Belleğin hex_value adresinde rd yazmacındaki değeri saklar
mov	`mov rd, rs1`	rd ← rs1
mvi	`mvi rd, [hex_value]h`	rd ← hex_value

2.2.5. Ek Komutlar

Mnemonik	Sentaks	Açıklama
cll	`cll`	Sistem çağrısı yapar.
dbs	`dbs name \”quoted_string\”`	Bellekte name adıyla referans verilebilen bir null-sonlandırmalı karakter dizisi kaydeder.

2.3. Mikroişlemci Çağrıları (Syscalls / Interrupts, Sistem Çağrıları / Kesmeler)

2.3.1. Sistemi Durdur (HALT)

Yazmaç	Beklenen Değer
x1	0
X2	İşlemin döndüreceği durum kodu

2.3.2. Ekrana Yazmaç Değeri Yazdır

Yazmaç	Beklenen Değer
x1	1
x2	Bu yazmaçtaki değer ekrana yazılacak.
x3	Sayı formatı (0: ikili; 1: onlu; 2: onaltılı; 3: utf8)

2.3.4. Ekrana String Yazdır

Yazmaç	Beklenen Değer
x1	2
x2	Stringin bellekteki adresi

2.3.5. Klavyeden Karakter Oku

Yazmaç	Beklenen Değer
x1	3
x2	Karakterin kaydedileceği bellek adresi

2.3.6. Klavyeden Girilen Stringi Oku

Okunan karakterin UTF-8 kodlama sisteminde numerik değerini **x3** yazmacındaki değere göre ya **x4** yazmacına kaydeder ya da **x2** yazmacında verilen bellek adresine kaydeder. Belleğe yazma durumunda hedef kayıt adresinden sonraki adrese **0** yazılacağına dikkat edilmelidir.

Yazmaç	Beklenen Değer
x1	4
x2	Karakterin kaydedileceği bellek bloğunun başlangıç adresi
x3	Kayıt konumunu seçer (0:**x4 y**azmacı; 1: **x2** yazmacındaki bellek adresi)

2.3.7. Klavyeden Sayı Oku

Verilen sayı kabul formatına göre okunan karakter dizisini sayıya çevirir ve **x3** yazmacında saklar. Eğer sayı verilen formatta okunamıyorsa **x4** yazmacına **1** değeri kaydedilir.

Yazmaç	Beklenen Değer
x1	5
x2	Sayı kabul formatı (0: ikili; 1: onlu: 2: onaltılık)

3. YÖNTEM

Komut seti simüle edilecek mikroişlemcinin seçilmesinde göz önünde bulundurduğumuz iki ana kriter var: Komut setinin basitliği ve güncel işlemci tasarımlarına benzerliği. Bu kriterleri göz önünde bulundurarak Zilog Z80, Konrad Zuse'un Z2'si, Intel C4004, 8086 ve 8088 detaylı inceledik. Son olarak da RISCV üzerinde inceleme yaptık ve bundan esinlenerek kendi mikroişlemcimiz olan "RISC-Mini"yi oluşturmaya karar verdik.

Geliştirilecek grafik arayüzlü ortamın temel bileşenlerini aşağıdakiler olarak belirledik:

1. Sanal işlemci çalışma zamanı motoru

2. İşlemcinin komutlarını okuduğu, kullanıcının çalışma zamanı motoru aracılığıyla ulaşabildiği sanal bir bellek

3. İşlemcinin yürütme süreci içerisinde veri kayıtlarını sakladığı genel amaçlı ve bayrak olmak üzere iki tür yazmaç

4. Kaynak kod editöründen yapılan girişleri ayrıştırarak (parsing) kontrol eden ve sanal belleğe yerleştirerek işlemcinin erişmesini sağlayan bir ayrıştırıcı (parser)

5. Kullanıcıdan Assembly dilinde kod kabul eden sonrasında bellek ve yazmaçtaki değişiklikleri gösteren bir kaynak editörü

Arayüzün tasarlanmasında geniş camialar tarafından kabul görmüş olan Figma aracının kullanılmasına karar verilmiş, Python ile GUI geliştirilmesinde ise en temel ve yaygın olan Tkinter modülü kullanım için seçilmiştir.

4. KURAMSAL TEMELLER ve KAYNAK ARAŞTIRMASI

Çalışmalarımıza başlamadan önce mikroişlemcinin gereksinimlerini ve neden kullanılması gerektiğini araştırarak başladık.

Mikroişlemciler, günümüzün bilgisayar ve elektronik sistemlerinde temel bir bileşen olarak kullanılırlar. Bu sistemler, mikroişlemci tarafından yürütülen komutlarla kontrol edilir ve yönetilir.

Bellek, işlemleri gerçekleştirmek için hafıza bloklarını bir listede saklamalıdır. Veri türü bellekte saklanacak verilerin sözcük uzunluğu boyutunda bellek alanında saklanmasını sağlar.

İşlemci mikroişlemcideki ana işlem görevlerini yürütür. Belleğe erişerek işlemleri bellek üzerinde gerçekleştirir. Mikroişlemci emirlerini yürütmek için yöntemlere sahiptir. İşlemci emirleri, örneğin, bellek adreslerini yüklemek, bellek adreslerine yazmak, sayısal işlemler yapmak, mantıksal işlemler yapmak, karşılaştırmalar yapmak ve atamalar yapmak gibi bir dizi işlemi içerebilir. İşlemci kodu çözerek hangi hangi işlemin yürütüleceğini belirlemelidir. Farklı kodlar için farklı işlemleri yürütmek için kullanılmalıdır. İşlemci gerçekleşen belirli işlemlerin verilerini geçici olarak saklamak için kayıtları temsil eden bir dizi içermelidir.

Bayraklar ise durumları ve hataları belirlemek için kullanılmalıdır. Bayraklar belirli işlemlerin yürütülmesinden sonra ayarlanır.

Kullanıcı arayüzü, mikroişlemci simülasyonunun kullanıcı tarafından etkileşimli olarak kontrol edilebilmesini sağlamak için kullanılmaktadır. Kullanıcı arayüzünde, kullanıcının kodu girebileceği bir metin kutusu bulunmaktadır. Buradaki kodun işlenmesi sonrasında bellekte, yazmaçlarda ve bayraklarda oluşan değişimleri takip edebileceğimiz bir alan bulunmaktadır. Bu alanları adım adım takip edebilmek hata ayıklama işlevselliğini sağlamaktadır.

5. SONUÇ (TARTIŞMA ve SONUÇ)

Sanallaştırılmış mikroişlemci yorumlama ortamları, birçok avantaj sunar. Bunlar arasında, farklı mikroişlemci mimarileri üzerinde çalışmanın kolaylığı, çeşitli ölçeklendirme seviyesinde çalışabilme yeteneği, verimli kod testi ve geliştirme olasılığı bulunmaktadır. Sanallaştırılmış mikroişlemci yorumlama ortamları, daha az donanım gereksinimiyle daha fazla sayıda sistem ve cihazın taklit edilebilmesine imkân sağlar.

Ancak, sanallaştırılmış mikroişlemci yorumlama ortamları da bazı dezavantajlara sahiptir. En büyük dezavantajı, gerçek sistemde olabilecek performans sorunlarının istenen şekilde taklit edilememesidir. Ayrıca, sanallaştırılmış mikroişlemci yorumlama ortamları, simülasyon teknolojilerinin birleştirilmesiyle oluşturulduklarından, sistem kaynaklarına daha fazla yük indirirler. Sonuç olarak, sanallaştırılmış mikroişlemci yorumlama ortamları, mikroişlemci tabanlı sistemlerin test edilmesi ve geliştirilmesi için önemli bir araçtır. Farklı özellikleri ve avantajları ile, kullanıcıların ihtiyaçlarına göre değerlendirilmelidir. Sanallaştırılmış mikroişlemci yorumlama ortamları, geliştirilmesi gereken yeni cihaz ve sistemlerin tasarım ve test süreçlerinde kullanılabilir. Ancak, gerçek sistem performansını tam olarak taklit edememe dezavantajı dikkate alınmalıdır.

6. KKM Üzerinde Çalışan Örnekler

6.1. Kullanıcıdan alınan 2 sayıyı toplayan program

```
;
;       Ferit Yigit BALABAN,    <fybalaban@fybx.dev>
;       RISC-Mini,      2023
;       toplama.asm

.global
    mvi x9, Ah ; x9 yazmacinda yeni satir karakterini tut

    mvi x1, 5h ; syscall no.5 hazirligi
    mvi x2, 1h ; sayiyi onlu sistemde oku
    cll        ; cagriyi gerceklestir
    mov x20, x3    ; okunan sayiyi x20'de sakla

    mvi x1, 2h ; syscall no.2 hazirligi
    mvi x2, [str0] ; ekrana yazilacak stringin adresini sakla
    cll        ; cagriyi gerceklestir

    mvi x1, 5h ; syscall no.5 hazirligi
    mvi x2, 1h ; sayiyi onlu sistemde oku
    cll        ; cagriyi gerceklestir
    mov x21, x3    ; okunan sayiyi x21'de sakla

    mvi x1, 2h ; syscall no.2 hazirligi
```

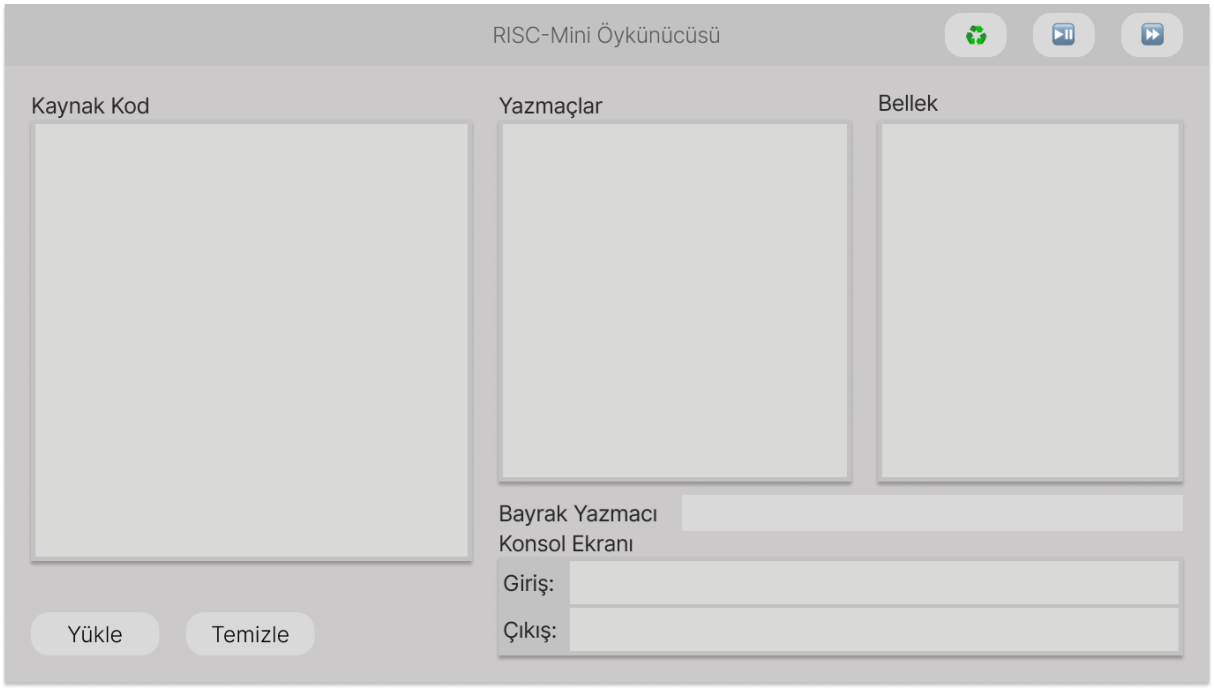
```
    mvi x2, [str1] ; ekrana yazilacak stringin adresini sakla
    cll       ; cagriyi gerceklestir

    mvi x1, 1h    ; syscall no.1 hazirligi
    add x2, x20, x21   ; ekrana yazilacak sayiyi (toplami) x2'de sakla
    mvi x3, 1h    ; sayiyi onlu sistemde yaz
    cll          ; cagriyi gerceklestir

    mvi x1, 2h ; syscall no.2 hazirligi
    mov x2, x9 ; yeni satir karakterini sakla
    cll       ; cagriyi gerceklestir

    mov x1, x0 ; islemciyi durdur
    mov x2, x0 ; durum kodu 0
    cll

.store
    dbs str0, " + "
    dbs str1, " = "
```

6.2. Kullanıcıdan adını alan ve ekrana yazan program

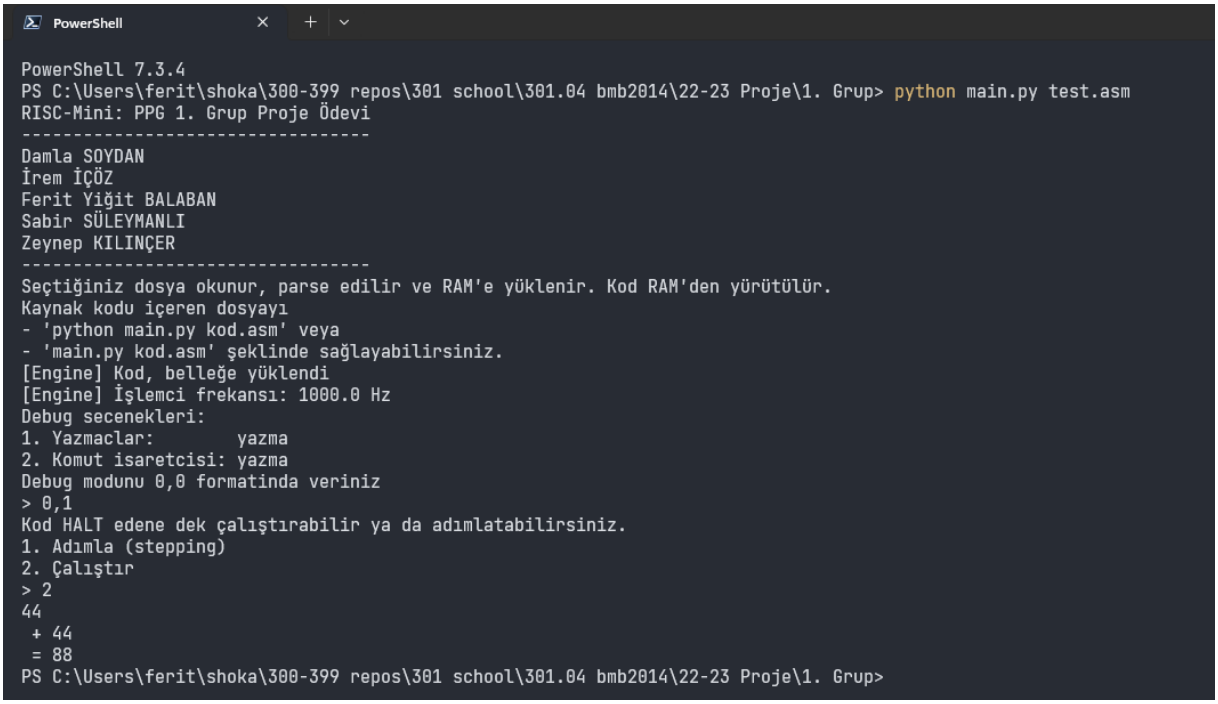
```
;
;       Ferit Yiğit BALABAN,    <fybalaban@fybx.dev>
;       RISC-Mini               2023
;       string_oku_ve_ekrana_yaz.asm

.global
    mvi x9, 10h     ; yeni satır karakterini x9'da sakla

    mvi x1, 2h
    mvi x2, [str0]
    cll

    mvi x1, 4h
    mvi x2, [FFh]
    cll

    mvi x1, 2h
    mvi x2, [str1]
    cll
    mvi x2, [FFh]
    cll
    mvi x2, [str2]
    cll
    mvi x1, 1h
    mov x2, x9
    mvi x3, 4h
    cll

    mov x1, x0
    mov x2, x0
    cll

.store
    dbs str0, "Adınızı girin: "
    dbs str1, "Merhaba, "
    dbs str2, "!"
```

7. Programdan Görseller

7.1. GUI (Grafik Arayüz) Tasarımı



7.2. TUI (Terminal Arayüzü) Üzerinde Çalışan RISC-Mini Instance'ı



8. KAYNAKLAR

1.https://www.eng.auburn.edu/~sylee/ee2220/8086_instruction_set.html

2.https://docs.python.org/3/library/tkinter.html

3.https://dev.to/yash_makan/4-ways-to-create-modern-gui-in-python-in-easiest-way-possible-5e0e

4.https://www.activestate.com/blog/top-10-python-gui-frameworks-compared/

5.https://en.wikipedia.org/wiki/Intel_8086

6.http://www.math.uaa.alaska.edu/~afkjm/cs221/handouts/irvine2.pdf

7.https://www.youtube.com/watch?v=Ps0JFsyX2fU

8.https://en.wikipedia.org/wiki/RISC-V

9.https://www.youtube.com/watch?v=Qd-jJjduWeQ

10.https://www.youtube.com/watch?v=66jIYW5kbj4

11.https://www.youtube.com/watch?v=7A_csP9drJw

12. https://www.cs.cornell.edu/courses/cs3410/2019sp/riscv/interpreter/

13. https://riscv.org/wp-content/uploads/2017/05/riscv-spec-v2.2.pdf

14. https://marz.utk.edu/my-courses/cosc230/book/example-risc-v-assembly-programs/

15. https://msyksphinz-self.github.io/riscv-isadoc/html/regs.html

16. https://msyksphinz-self.github.io/riscv-isadoc/html/rvi.html

17. https://itnext.io/risc-v-instruction-set-cheatsheet-70961b4bbe8?gi=a48779e4d7eb

18. https://www.pcpolytechnic.com/it/ppt/8086_instruction_set.pdf

19. https://web.karabuk.edu.tr/emelkocak/indir/MTM305/KOMUT%20SET%C4%B0.pdf

20. https://aakgul.sakarya.edu.tr/sites/aakgul.sakarya.edu.tr/file/_8086KomutlarOrnekler.PDF

21. https://en.wikipedia.org/wiki/MOS_Technology_6508

22. https://en.wikipedia.org/wiki/Intel_4004

23. https://en.wikipedia.org/wiki/Zilog_Z80

24. https://en.wikipedia.org/wiki/Z1_(computer)

25. https://www.youtube.com/watch?v=cNN_tTXABUA

26. https://www.youtube.com/watch?v=Z5JC9Ve1sfI

27. https://www.youtube.com/watch?v=sK-49uz3lGg

28. https://www.youtube.com/watch?v=QXjU9qTsYCc

29. https://github.com/fybx/processor

30. https://github.com/fybx/interpreter