KAYAN NOKTALI SAYILAR

Amaç: IEEE 16-bit formatındaki iki adet kayan nokta sayının toplama programının assembly dilinde yazımı

Giriş: Morris Mano komutları ve bu bilgisayara ait assembly dili

HAZIRLAYAN:

Ad Soyad: Zekeriya Bilgin

İÇİNDEKİLER

GİRİŞ - ÖNSÖZ **2**

KAYAN NOKTA VE IEEE754 **3**

AKIŞ ŞEMASI **4**

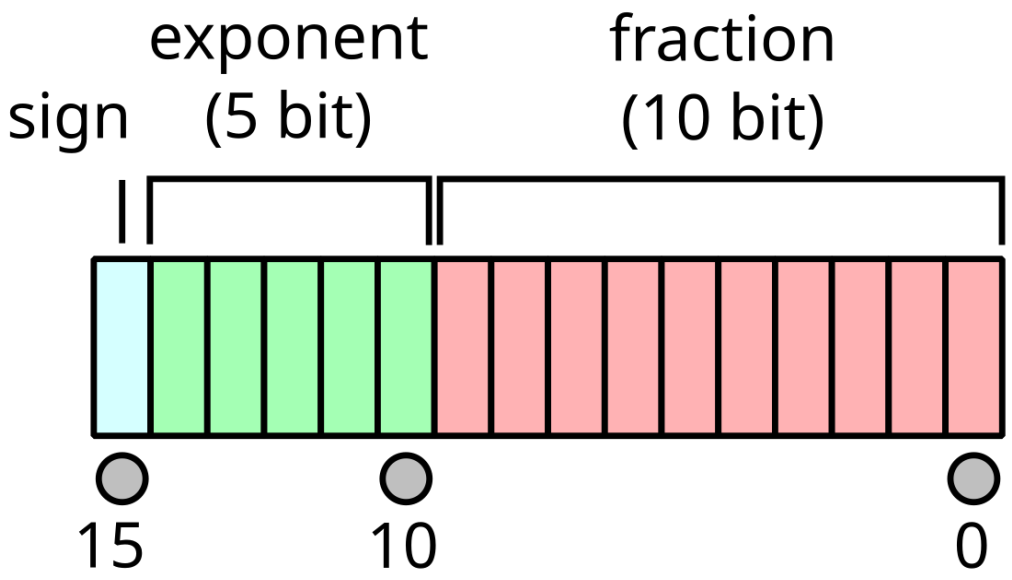
KODLAR VE AÇIKLAMALARI **8**

YORUM – SON **15**

-ÖNSÖZ-

Projeye başlarken bizden istenileni net bir şekilde kavrayalım. Elimiz de iki tane IEEE 16- bit floating point formatında sayımız var. Bu iki sayının toplama programını Morris Mano' nun bilgisayarının komutları ve bu dile ait assembly dili kullanarak gerçekleştireceğiz.

IEEE 754 16-Bit formatında kayan noktalı sayı gösterim biçimi şu formattadır.



15. bitimiz İşaret (Sign) bitimiz.

14-10 arasında ki 5 bitimiz Üst (Exponent) bitlerimiz.

9-0 arasında ki 10 bitimiz ise Mantisa/Kesit(Fraction) bitlerimiz.

IEEE 754 -16 Bit formatı Half-precision floating-point format olarak yani Yarım duyarlıklı kayan noktalı sayı biçimi olarak adlandırılır.

Bu format, özellikle belleğin sınırlı olduğu veya yüksek performansın kritik olduğu uygulamalarda kullanılır. Yarım duyarlıklı kayan nokta formatı, tam duyarlıklı (32 bit) veya çift duyarlıklı (64 bit) formatlara kıyasla daha az hassasiyet ve dinamik aralık sunar, ancak daha az bellek kullanır ve daha hızlı işlem yapılabilir.

1. İşaret biti (1 bit):

- 0: Pozitif sayılar
- 1: Negatif sayılar

2. Üs (5 bit):

- Bias: 2^(5-1) - 1 = 15
- Bias: 2^(Üst bit sayısı-1)= 15
- Üs değeri: [0, 31]
    - Üs = 0 ve mantissa ≠ 0: Denormalize edilmiş sayı (subnormal)

- Üs = 0 ve mantissa = 0: Sıfır (0)
- Üs = 31 ve mantissa = 0: Sonsuzluk (∞ veya -∞)
- Üs = 31 ve mantissa ≠ 0: NaN (Not a Number)
- Diğer durumlar: Normalize edilmiş sayı

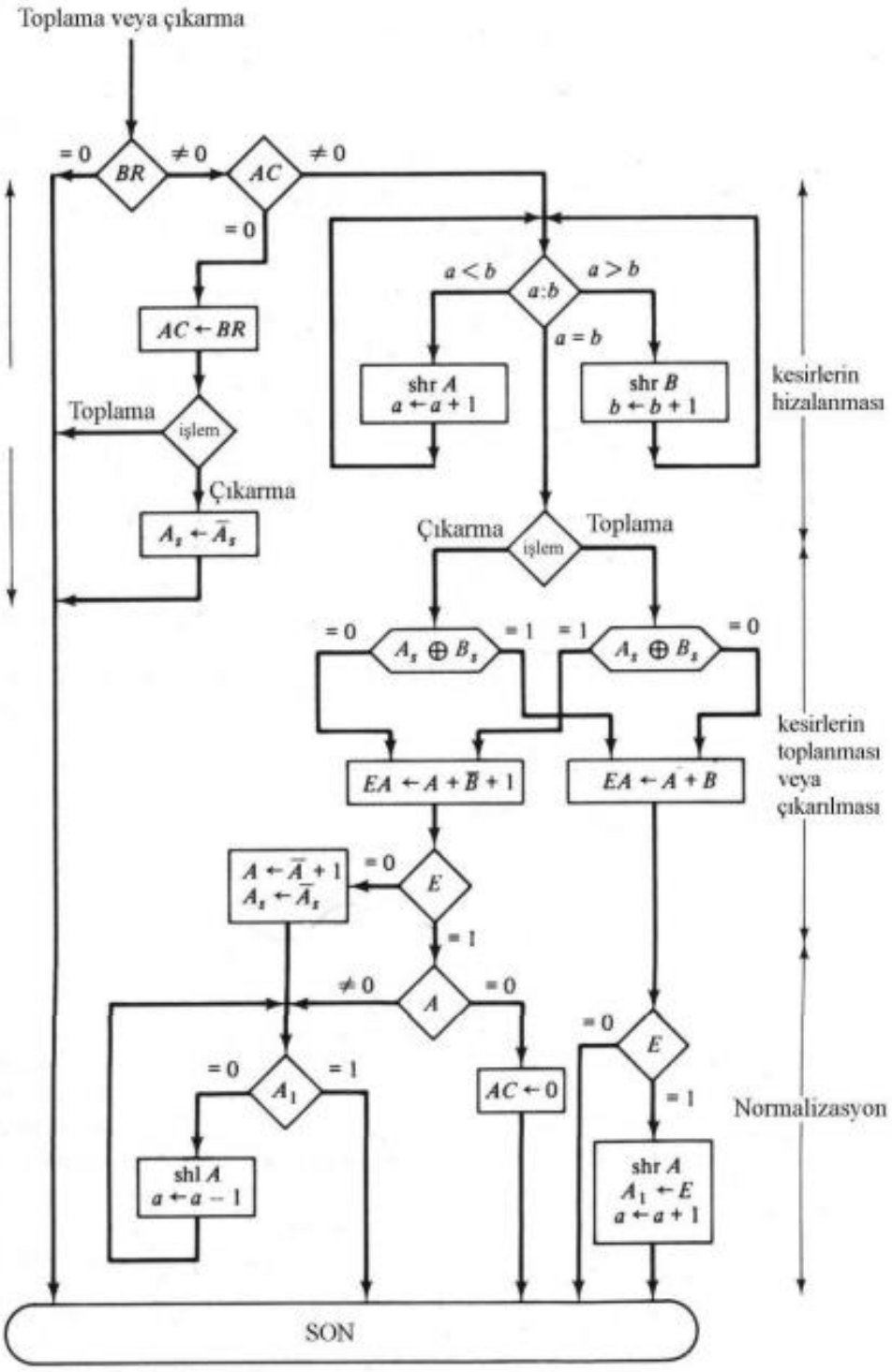
3. Mantissa (10 bit):

- Mantissa (veya "kesir" olarak da adlandırılır) 1 bit gizli (implicit) öndeki 1 ile normalize edilir. Bu nedenle, sadece 10 bitlik kesir kısıtlanır.
- Normalize edilmiş sayılar için, mantissa'nın bu formatı gizli 1 ile başlar.

-AKIŞ ŞEMAMIZ-

Morris Mano nun Bilgisayar Sistemleri Mimarisi(Computer System Architecture) kitabını incelediğimizde sayfa 337 aşağıda ki resim de görünen 'Kayan noktalı sayıların toplanması ve çıkarılması' şemasının toplama adımlarını kullanabileceğimizi görüyoruz. İstersek yine kendimiz de farklı akış semaları veya yöntemler kullanabiliriz. Ben kitaba bağlı kalmak adına bu akış şemasında ki yöntem ile devam ettim.

Yukarıda bahsettiğim demormalize olan normalize edilemeyen durumları akış şemasında ki gibi özel olarak değerlendirdim.



Şekil 10.15 Kayan noktalı sayıların toplanması ve çıkarılması

Bu akış şemasın da sadeleştirme yapıp çıkarma işlemi adımlarını işlemeyeceğimiz için es geçiyoruz.

Programımızın genel akışı şu şekilde olacaktır;

1. Sıfırların denetimi

2. Kesirlerin hizalanması

3. Kesirlerin toplanması

4. Sonucun normalizasyonu

Tabii bir hususu tekrardan hatırlatmakta fayda var. Kayan noktalı sıfır normalize edilemez. Normalize sırasında sıfır denetimi yapmak yerine gerekirse işlemin başlangıcında ve sonunda denetim yapılabilir. Kesirlerin hizalanması, onların işleme girmesinden önce yapılmalıdır. Kesirlerin toplanmasından sonra sonuç normalize almayabilir. Normalize işlemiyle, belleğe gidecek verinin belleğe gitmeden önce normalize edilmiş olması sağlanır.

Eğer sayı1 sıfıra eşitse işlem sona erer. Sayı2 sonuçtur. Eğer sayı2 sıfır ise, sayı1 sonuçtur. Eğer sayıların hiçbiri sıfır değil ise sayıların hizalanmasına geçilir.

Morris Mano'nun bilgisayarının 25 adetten oluşan komutlarına tekrardan göz atalım.

Çizelge 5.6 Temel bilgisayar için denetim fonksiyonu ve Mikro işlemler.

Al-getir	$R'T_0$:	$AR \leftarrow PC$
	$R'T_1$:	$IR \leftarrow M[AR]$, $PC \leftarrow PC + 1$
Kodunu çöz	$R'T_2$:	$D_0, \ldots, D_7 \leftarrow$ kodunu çöz IR (12–14) $AR \leftarrow IR$ 0–11), $I \leftarrow IR(15)$
Dolaylı	$D'7IT_3$:	$AR \leftarrow M[AR]$
Kesme:		
$T'_0T'_1T'_2(IEN)(FGI + FGO)$:		$R \leftarrow 1$
	RT_0:	$AR \leftarrow 0$, $TR \leftarrow$ PC
	RT_1:	$M[AR] \leftarrow TR$, $PC \leftarrow 0$
	RT_2:	$PC \leftarrow PC + 1$, $IEN \leftarrow 0$, $R \leftarrow 0$, $SC \leftarrow 0$
Bellek adreslemeli buyruklar		
ADD	D_0T_4:	$DR \leftarrow M[AR]$
	D_0T_5:	$AC \leftarrow AC \wedge DR$, $SC \leftarrow 0$
	D_1T_4:	$DR \leftarrow M[AR]$
	D_1T_5:	$AC + DR$, $E \leftarrow C_{çıkış}$, $SC \leftarrow 0$
LDA	D_2T_4:	$DR \leftarrow M[AR]$
	D_2T_5:	$AC \leftarrow DR$, $SC \leftarrow 0$
STA	D_3T_4:	$M[AR] \leftarrow AC$, $SC \leftarrow 0$
BUN	D_4T_4:	$PC \leftarrow AR$, $SC \leftarrow 0$
BSA	D_5T_4:	$M[AR] \leftarrow PC$, $AR \leftarrow AR + 1$
	D_5T_5:	$PC \leftarrow AR$, $SC \leftarrow 0$
ISZ	D_6T_4:	$DR \leftarrow M[AR]$
	D_6T_5:	$DR \leftarrow DR + 1$
	D_6T_6:	$M[AR] \leftarrow DR$, eğer $(DR = 0)$ ise $(PC \leftarrow PC + 1)$, $SC \leftarrow 0$
Yazaç adreslemeli buyruklar		
	$D_7IT_3 = r$ (tüm yazaç adreslemeli buyruklarda ortak)	
	$IR(i) = B_i$ $(i = 0, 1, 2, \ldots, 11)$	
	r:	$SC \leftarrow 0$
CLA	rB_{11}:	$AC \leftarrow 0$
CLE	rB_{10}:	$E \leftarrow 0$
CMA	rB_9:	$AC \leftarrow \overline{AC}$
CME	rB_8:	$E \leftarrow \overline{E}$
CIR	rB_7:	AC Shr AC, $AC(15) \leftarrow E$, $E \leftarrow AC(0)$
CIL	rB_6:	AC Shl AC, $AC(0) \leftarrow E$, $E \leftarrow AC(15)$
INC	rB_5:	$AC \leftarrow AC + 1$
SPA	rB_4:	Eğer $(AC(15) = 0)$ ise $(PC \leftarrow PC + 1)$
SNA	rB_3:	Eğer $(AC(15) = 1)$ ise $(PC \leftarrow PC + 1)$
SZA	rB_2:	Eğer $(AC = 0)$ ise $(PC \leftarrow PC + 1)$
SZE	rB_1:	Eğer $(E = 0)$ ise $(PC \leftarrow PC + 1)$
HLT	rB_0:	$S \leftarrow 0$
Giriş-çıkış buyrukları		
	$D_7IT_3 = p$ (tüm giriş-çıkış buyruklarında ortak)	
	$IR(i) = B_i (i = 6, 7, 8, 9, 10, 11)$	
	p:	$SC \leftarrow 0$
INP	pB_{11}:	$AC(0–7) \leftarrow INPR$, $FGI \leftarrow 0$
OUT	pB_{10}:	$OUTR \leftarrow AC(0–7)$, $FGO \leftarrow 0$
SKI	pB_9:	Eğer $(FGI = 1)$ ise $(PC \leftarrow PC + 1)$
SKO	pB_8:	Eğer $(FGO = 1)$ ise $(PC \leftarrow PC + 1)$
ION	pB_7:	$IEN \leftarrow 1$
IOF	pB_6:	$IEN \leftarrow 0$

Basit ve sayısı az en temel işlemler için gerekli komutların olduğunu görüyoruz. Bunların farklarını birazdan daha geniş bir komut seti ile kıyaslandığında nasıl bir farka sebep olacağına ileride değineceğiz.

-BELLEK-

-Notasyon: Kodun okunabilirliğini arttırmak ve anlaşılabilirliğini kolaylaştırmak adına etiket(label) ve sembolik adresleri(komutun yanındaki) Türkçe ve uzun kelimeler ile kullandım. Standartta sembolik bir adresinin 1. Karakterin harf olmak zorunda olduğunu ve en fazla 3 alfanümerik karakterden oluştuğunu unutmayın.

Bellek de kullanacağımız alanlar, bunların sembolik adreslerini ve içeriklerini görelim.

ETİKET	BELLEK ADRESİ	İÇERİK
SAYIA	100	0101001000000000
SAYIB	101	1100000100000000
MASKEİŞARET	102	1000000000000000
MASKEÜST	103	0111110000000000
MASKEMANTİS	104	0000001111111111
MASKEMANTİSMSB	105	0000001000000000
XOR	106	0000000000000000
SAYIAİŞARET	107	0000000000000000
SAYIBİŞARET	108	0000000000000000
SAYIAÜST	109	0000000000000000
SAYIBÜST	110	0000000000000000
FARKSAYISI	111	0000000000000000
SONUCİŞARET	112	0000000000000000
SONUCÜST	113	0000000000000000
SONUC	114	0000000000000000
AZALT	115	1011110000000000
ARTTIRMANTİS	116	0011110000000000
ARTTIRÜST	117	0000010000000000
MASKESIFIR	118	1000000000000000

Tabii ben burada anlaşılması kolay olması adına çok fazla sayıda bellek gözü kullandım. Programın tasarlanması ve organizasyonuna bağlı olarak sonuç aynı adres gözüne yazılabilir(Sonucun sayıA üzerine yazılması gibi) ve diğer optimizasyonlar yapılabilir.

NOTASYON VE FORMAT

A = Sayı A'nın kesir-mantis kısmı.

B = Sayı B'nin kesir-mantis kısmı.

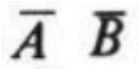
a = Sayı A'nın üst-exponent kısmı.

b = Sayı B'nin üst-exponent kısmı.

As = Sayı A'nın işaret biti. Bs = Sayı B'nin işaret biti.

A1 = Kesrin-Mantisanın En Büyük Değerli Basamak biti(Most SignificantDigit)

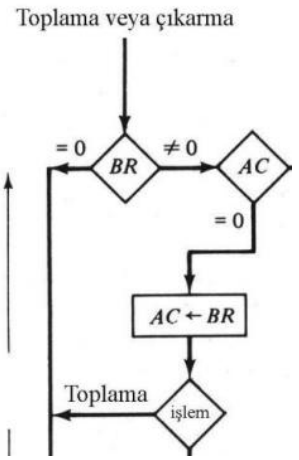
⊕ = Xor Özel Veya (eXclusive OR) Kapısı

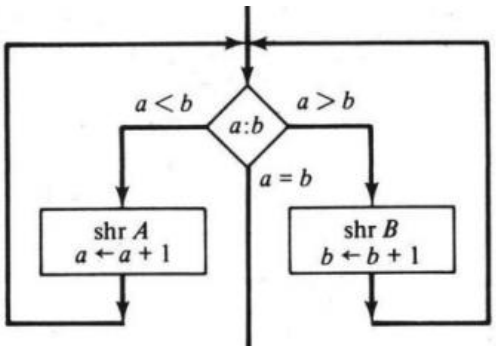
 A nın ve B nin tümleyenleri(Değilleri)

-PROGRAM BAŞLANGICI-

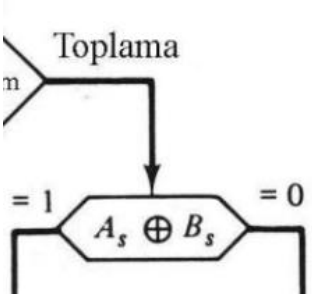
```
ORG 118                    Programımız 118 nolu bellek
---------------           adresinden çalışmaya başladı.
LDA SAYIA                  Burada sayı a nın ve
AND MASKEİŞARET            sayı b nin 3 kısma
STA SAYIAİŞARET            (işaret-üst-mantis)
LDA SAYIB                  bölünerek gerekli
AND MASKEİŞARET            bitlerinin maskelenemsini
STA SAYIBİŞARET            ardından kaydedilmesini
---------------            sağladım.
LDA SAYIA
AND MASKEÜST
STA SAYIAÜST
LDA SAYIB
AND MASKEÜST
STA SAYIBÜST
--------------
LDA SAYIA
AND MASKEMANTİS
STA SAYIAMANTİS
LDA SAYIB
AND MASKEMANTİS
STA SAYIBMANTİS
---------------
```

```
                    LDA SAYIA
                    AND MASKESIFIR                      Kayıtlardan sonra
                    SZA (SAYIA= 0 İSE SONUÇ SAYIB)      sıfır kontrolü
                    BUN SayıA!=0
                    LDA SAYIB
                    STA SONUC
                    HLT
SayıA!=0,           LDA SAYIB
                    AND MASKESIFIR
                    SZA (SAYIA !=0 V SAYIB = 0 SONUÇ SAYIA)
                    BUN  A !=0 V B !=0
                    LDA SAYIA
                    STA SONUÇ
                    HLT
-----------------------------------------
sayıA!=0sayıVB!=0   LDA SAYIBÜST                        İki sayıda sıfır değil
                    CMA                                 üstlerin kontrolü ve
                    INC                                 hizalanması.
                    ADD SAYIBÜST
                    SZA (a=b)
                    BUN a != b
                    BUN İŞARETEBAK
a != b              SPA (a-b pozitif a>b)
                    BUN b>a
                    CLE
                    CIL
                    CIL
                    CIL
                    CIL
                    CIL
                    CIL
                    CIL
```





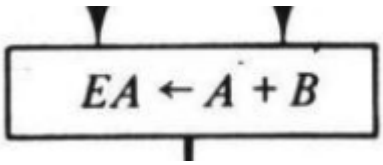
```
                STA FARKSAYISI          üstler arasında ki farkın
DÖNGÜ1          SZA (AC=0 ?)            bulunup belleğe kaydedilip
                BUN  FARKAZALT          bir döngü kurulması
                BUN FARKBİTTİ
FARKAZALT       ADD EKSİ1
                STA FARKSAYISI
                LDA SAYIBMANTİS
                CIR
                CLE
                STA SAYIBMANTİS
                LDA SAYIBÜST
                ADD USTARTTIR(0400)
                STA SAYIBÜST
                LDA FARKSAYISI
                BUN DÖNGÜ1
FARKBİTTİ       LDA SAYIBÜST
                AND MASKÜST
                STA SONUCUST
                BUN İŞARETEBAK
b>a             CMA
                INC
                AND MASKEÜST      ?
                STA FARKSAYISI
DÖNGÜ2          SZA (AC == 0 ?)
                BUN FARKAZALT2
                BUN FARKBİTTİ2
FARKAZALT2      ADD EKSİ1
                STA FARKSAYISI
                LDA SAYIAMANTİS
                CIR
                CLE
                STA SAYIAMANTİS
                LDA SAYIAÜST
```

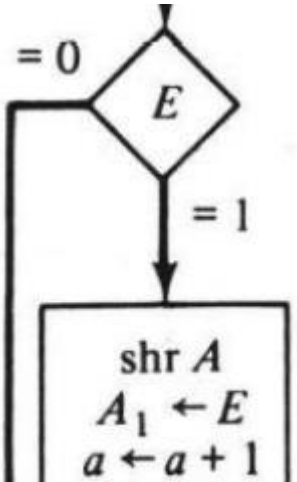


```
                ADD SAYIUSTARTTIR(0400)
                STA SAYIAÜST
                LDA FARKSAYISI
                BUN DÖNGÜ2
FARKBİTTİ2      LDA SAYIAÜST
                AND MASKEÜST                    XOR= (A'.B)+(A.B')
İŞARETEBAK      LDA SAYIAİŞARET          Sayıların işaretlerine
                CMA                     xor ile bakılması manoda
                AND SAYIBİŞARET          xor komutu olmadığı için
                STA XOR                 dolaylı olarak-diğer komutlar
                LDA SAYIBİŞARET          vasıtasıyla yapılması
                CMA
                AND SAYIAİŞARET
                ADD XOR
                SPA ((As Exor Bs) = 0 işaretleri aynı)
                BUN FARKLIİŞARET
                LDA SAYIAMANTİS
                ADD SAYIBMANTİS
                CIL
                CIL                             Aynı işaret olması durumu
                CIL
                CIL
                CIL
                CIL
                SZE (E=0)
                BUN E=1                          Elde biti(E) nin
                CIR                             içeriğine bağlı oluşan
                CIR                             durumlar.
                CIR
                CIR
                CIR
                CIR
                ADD SONUCÜST
```

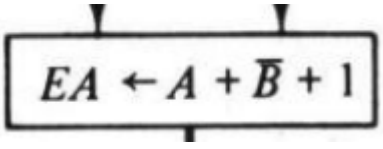
2 input XOR gate

A	B	A⊕B
0	0	0
0	1	1
1	0	1
1	1	0

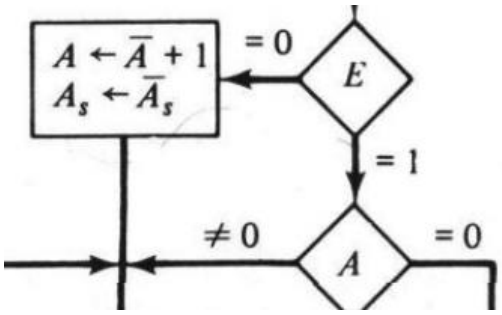


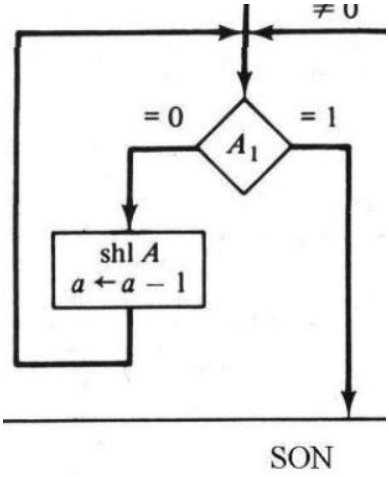


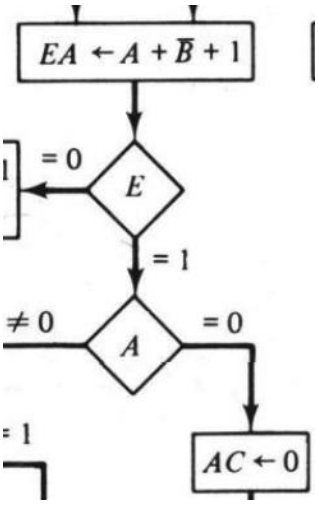
```
                ADD SAYIAİŞARET
                STA SONUC
                HLT
E = 1           CIR
                CIR
                CIR
                CIR
                CIR
                CIR
                CIR
                CIR
                CLE
                CIR
                CLE
                STA SONUCMANTİS
                LDA SONUCÜST
                ADD ARTTIRÜST(0400)
                AND MASKEÜST
                ADD SONUCMANTİS
                ADD SAYIAİŞARET
                STA SONUC
                HLT
FARKLIİŞARET    LDA SAYIBMANTİS          Farklı işaretler olması
                CMA                      durumunda yapılacak işlem
                INC
                ADD SAYIAMANTİS
                STA SONUCMANTİS
                CIL
                CIL
                CIL
                CIL
                CIL
                CIL
                SZE(E = 0 )              Elde biti kontrolü
```

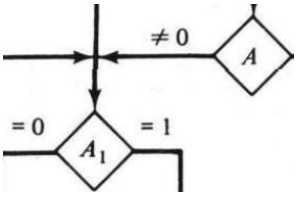


```
                BUN 2E=1
                LDA SONUCMANTİS
                CMA
                INC
                STA SONUCMANTİS
                LDA SAYIAİŞARET
                CMA
                STA SAYIAİŞARET
A1KARAR         LDA SONUCMANTİS
                AND MASKEMANTİSMSB(0200)
DÖNGÜ3          SZA (AC = 0 yani mantis msb =0 )
                BUN MSBMANTİS=1
                LDA SONUCMANTİS
                CLE
                CIL
                CLE
                STA SONUCMANTİS
                LDA SONUCUST
                ADD EKSİ1
                AND MASKEÜST
                STA SONUCÜST
                ADD SONUCMANTİS
                ADD SONUCİŞARET
                STA SONUC
                BUN DÖNGÜ3
MSBMANTİS=1     HLT
2E=1            CIR
                CIR
                CIR
                CIR
                CIR
                CIR
                SZA (MANTİS=0)
```









```
                BUN MANTİS!=0
                CLA
                STA SONUC
                HLT
MANTİS!=0       BUN A1KARAR
```

-YORUM-

RISC-V komut listesinde yer alan SUB komutu Mano komut listesinde olsaydı çıkartma işlemi olarak kullandığımız 2 ye tümleyen aritmetiği ile yaptığım bu işlemi çok rahatlıkla yapardım.

Örnek;

----RISC-V Komutları----		----Mano Komutları----
SUB RD, RS1, RS2	=	LDA RS2
		CMA
		INC
		ADD RS1
		STA RD

Yine aynı şekilde karşılaştırma ve xor işlemleri mano komutları ile direkt olarak yapılamamaktadır. RISC-V komut listesinde yer alan XOR komutu Mano komut listesinde olsaydı (A'.B)+(A.B') formatında yaptığım bu işlemi çok rahatlıkla yapardım.

Örnek;

----RISC-V Komutları----		----Mano Komutları----
XOR RD, RS1, RS2	=	LDA RS2
		CMA
		AND RS1
		STA GEÇİCİ
		LDA RS1
		CMA
		AND RS2
		ADD GEÇİCİ

SLTU - Set Less Than Unsigned (İşaretsiz) Bu komut, işaretsiz karşılaştırma yapar. rs1 işaretsiz olarak rs2'den küçükse, rd kaydına 1 yazılır, aksi halde 0 yazılır. SLTU komutu Mano komut listesinde olsaydı aşağıdaki örnekte yaptığım bu işlemi çok rahatlıkla yapardım.

*sayılar işaretsiz olarak ele alınmıştır. İşaretli olanlar için slt komutu kullanınız.

----RISC-V Komutları----		----Mano Komutları----
SLTU RD, RS1, RS2	=	LDA RS2
		CMA
		INC
		ADD RS1
		SPA
		BUN 1
		BUN 0

Örneklerden de anlaşıldığı gibi, özel VEYA (XOR), çıkarma (SUB) ve karşılaştırma (SLTU) gibi en çok kullanılan işlemler, RISC-V komut setinde doğrudan desteklenirken, Mano bilgisayar komutları arasında bulunmamaktadır. Bu durum, RISC-V komut seti kullanıldığında tek bir komut ile yapılabilen işlemlerin, Mano'da birden fazla komut kullanılarak gerçekleştirilmesini gerektirmektedir. Örneğin, RISC-V komutlarıyla tek 1 komut ile yapılabilecek bir işlem, Mano'da 8 komut ile yapılmaktadır.

Bu fark, bir işlemcinin ve bilgisayarın kullandığı komut setinin ne denli büyük ve önemli bir etkiye sahip olduğunu açıkça göstermektedir. İşlemci tasarımı ve organizasyonu yapılırken, hedef uygulamaların gereksinimlerine ve kullanım alanlarına uygun komut seti tasarımı yapılması gerekmektedir. Gelişmiş işlemciler, geniş ve kapsamlı komut setleri sunarak yüksek performans ve esneklik sağlarken, basit işlemciler daha sınırlı komut setleri ile eğitim ve belirli uygulama gereksinimlerine hizmet etmektedir.

Bilgisayar teknolojilerindeki gelişmeler, işlemci mimarilerinin de sürekli olarak evrilmesine neden olmuştur. RISC-V gibi açık kaynaklı mimariler, bu evrimde önemli bir rol oynamaktadır.

Mano gibi basit komut setleri, eğitim amaçlı ve küçük ölçekli uygulamalar için uygun olabilir. Ancak, büyük ölçekli ve karmaşık uygulamalar için RISC-V gibi daha gelişmiş mimariler tercih edilir.

Okuduğunuz için teşekkürler.

Temel Kaynak : Morris Mano Computer System Architecture