ELM 234 - Ödev #4
Berat KIZILARMUT
171024086

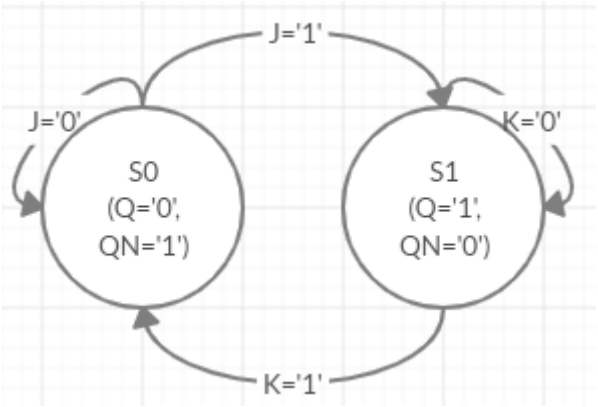
A. FSM Devre Tasarımı

---

1. FSM kullanarak JK Flip Flop çalışma prensibine uygun devre tasarlayın.

J	K	clk	Q
0	0	1	Önceki konum
0	1	1	0
1	0	1	1
1	1	1	toggle

Yandaki Truth table'a sahip olan bir JK Flip Flop devresinin SFM gösterimi;




```
module odev4_a1(
input logic J, K, clk,
output logic Q, QN
);
typedef enum {S0, S1} statetype;
statetype state, nextstate;
always_ff @(posedge clk)
begin
    state <= nextstate;
end

always_comb
begin
    case(state)
    S0:
    begin
        if (J)
            nextstate = S1;
        else
            nextstate = S0;
    end
    S1:
    begin
        if(K)
            nextstate = S0;
        else
            nextstate = S1;
    end
    default: nextstate = S0;
    endcase

    case(state)
    S0:
    begin
        Q = 0;
        QN = 1;
    end
    S1:
    begin
        Q = 1;
        QN = 0;
    end
    endcase
endmodule
```

2. Şekil 1 de verilen diagrama uygun devre için;

a. Statelerinizi binary kodlaması ile seçerek devre tasarlayın ve son devreyi çizin

Binary kodlaması ile, S0, S1 ve S2 stateleri, sırası ile 00, 01 ve 10 olur.

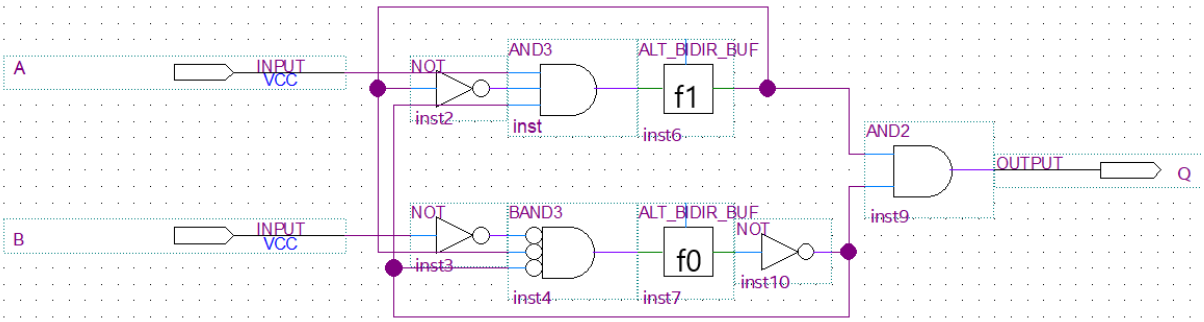
Belirlediğimiz state tanımlarına göre elimizdeki FSM Şemasını oturtuyoruz ve next stateleri buluyoruz.

Current State		Inputs		Next State	
f0	**f1**	**A**	**B**	**f'0**	**f'1**
0	0	1	X	1	0
1	0	X	1	0	1
1	0	X	0	0	0
0	1	X	X	0	0

$$f_1' = B * \overline{f_1} * f_0, \qquad f_0' = A * \overline{f_1} * \overline{f_0}$$

S0 (00), S1 (01) ve S2 (10) statelerinin çıkışları sırasıyla Q=0, Q=0 ve Q=1 olmaktadır.

$$Q = f_1 * \overline{f_0}$$



b. Statelerinizi one-hot kodlaması ile seçerek devre tasarlayın ve son devreyi çizin

One-hot kodlaması ile, S0, S1 ve S2 stateleri, sırası ile 001, 010 ve 100 olur.

Current State			Inputs		Next State		
f0	**f1**	**f2**	**A**	**B**	**f'0**	**f'1**	**f'2**
0	0	1	X	X	1	0	0
0	1	0	X	1	0	0	1
0	1	0	X	0	1	0	0
1	0	0	1	X	0	1	0

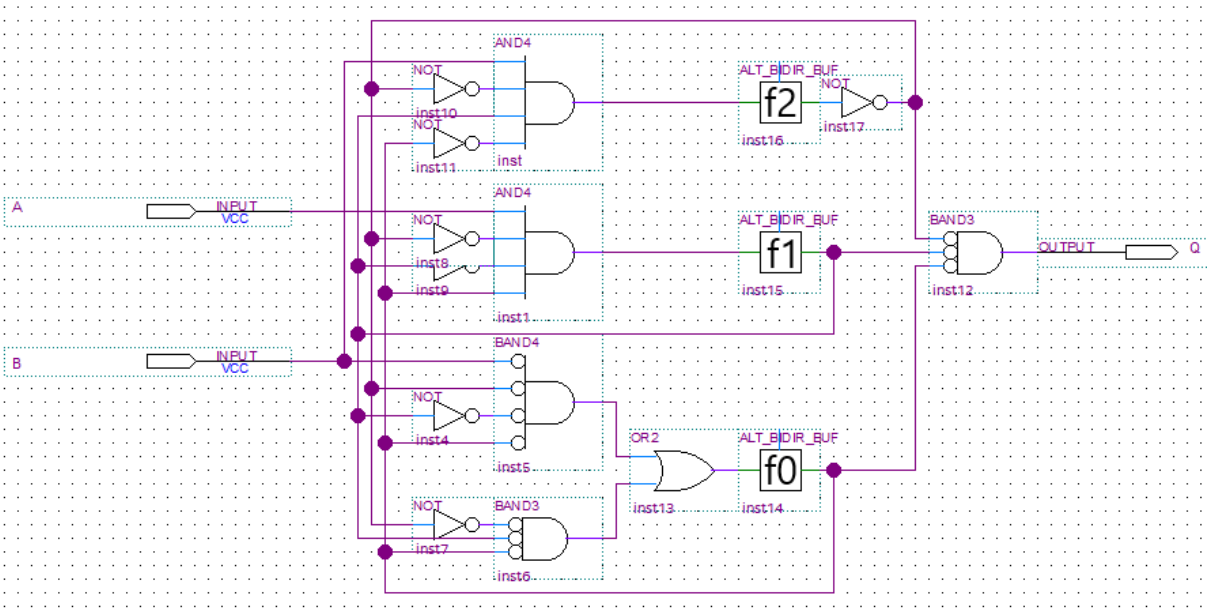
$$f'_0 = B * \overline{f_0} * f_1 * \overline{f_2} + \overline{f_0} * \overline{f_1} * f_2$$

$$f'_1 = A * f_0 * \overline{f_1} * \overline{f_2}$$

$$f'_2 = B * \overline{f_0} * f_1 * \overline{f_2}$$

S0 (001), S1 (010) ve S2 (100) statelerinin çıkışları sırasıyla Q=0, Q=0 ve Q=1 olmaktadır.

$$Q = \overline{f_0} * \overline{f_1} * f_2$$

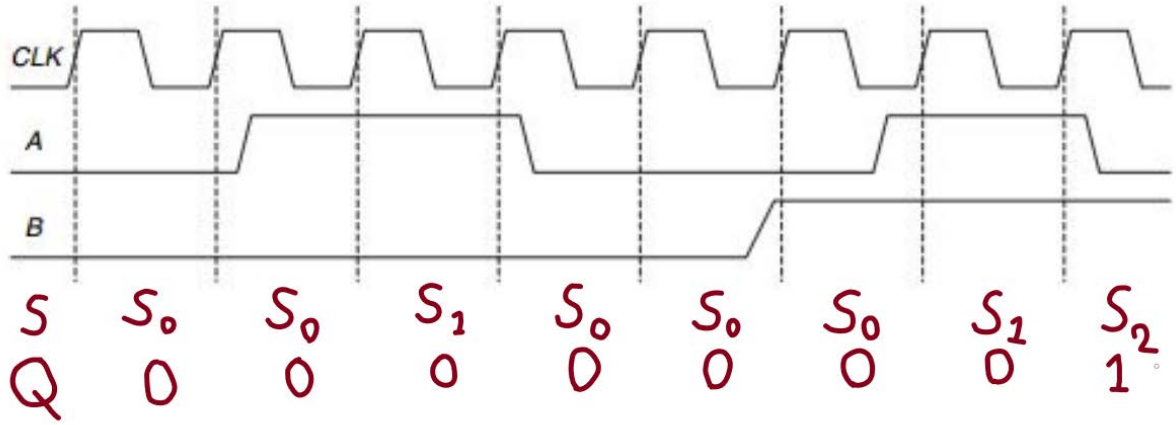


c. a ve b şıklarındaki devreler arasındaki farkları yorumlayın.

One-hot kodlaması yaptığımızda devremiz daha karışık duruma gelmektedir. Statelerimiz üçer bit hale gelmektedir. Binary kodlamasında ise statelerimiz ikişer bitliktir ve devre daha basittir. Bir register daha az gerekmektedir.

3. Şekil 1 deki devreye, (ilk clock gelmeden devrenin resetlendiğini varsayarak) Şekil 2 deki gibi giriş uygulandığında, devre hangi state de olur ve Q çıkışının değeri nedir?

Devrenin Trigger'ını Clock Rising Edge olarak varsaydığımızda, S ve Q sinyallerimiz aşama aşama şekildeki gibi olacaktır.

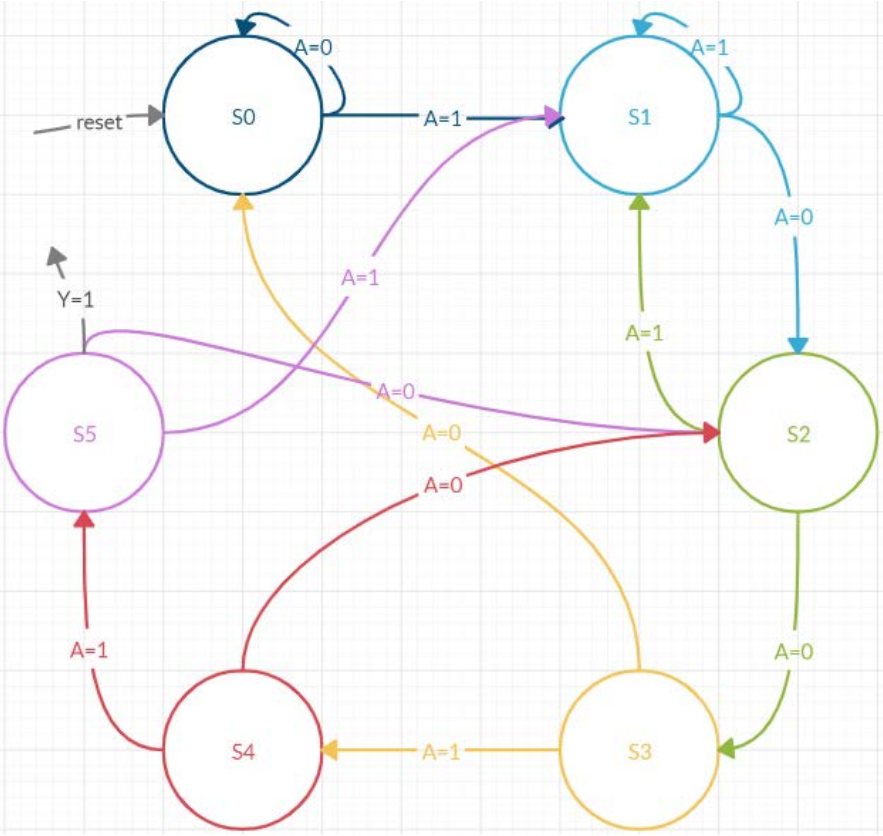


B. Pattern Tanıma

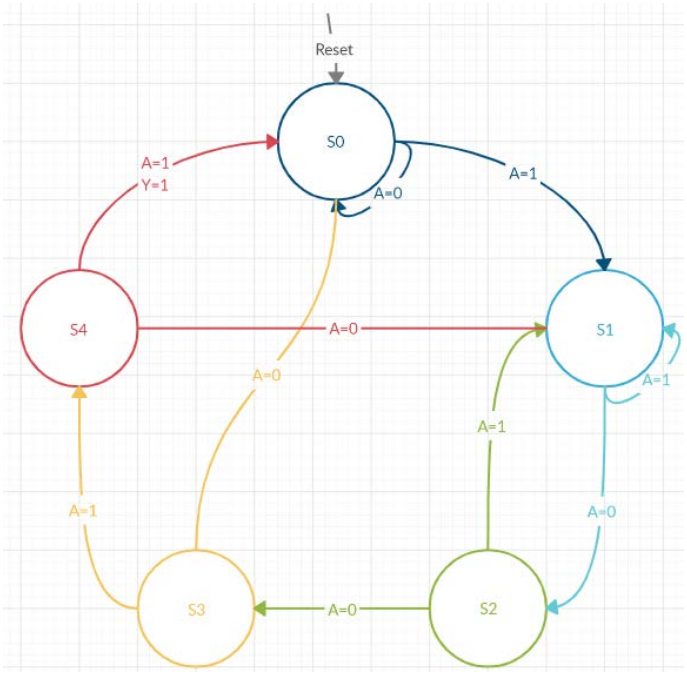
---

4. A girişinden her clock sinyalinde gelen bitlerden 10011 patternini yakaladığında X çıkışını

a. Moore State Transition Diagramını çizin.



b. Mealy State Transition Diagramını çizin.



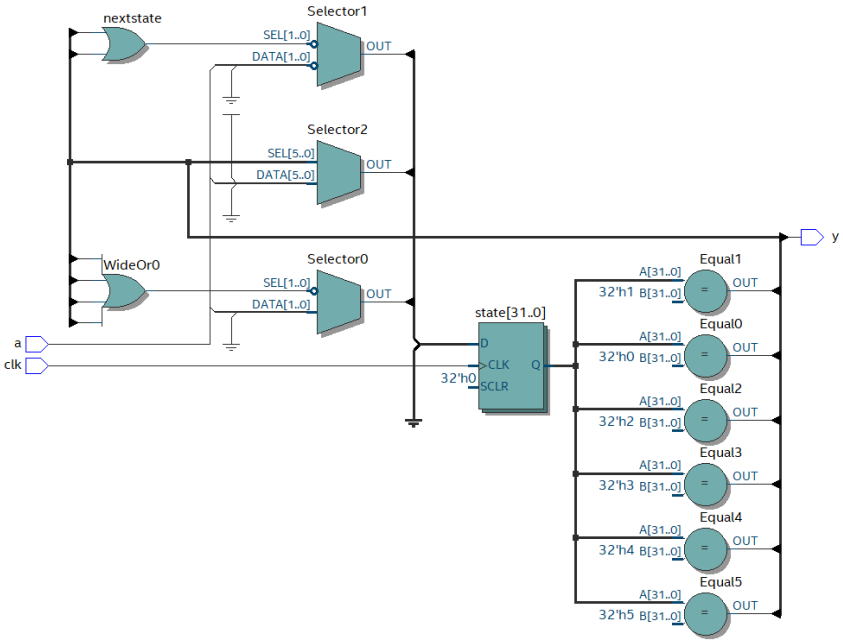
c. a veya b şıkkındaki diagramınızdan istediğiniz bir tanesini baz alarak FSM devresini oluşturun. Statelerinizi Johnson kodlaması ile kodlayınız.

Moore Devresi seçildi ve oluşturuldu.

```
module odev4_b4(
    input logic a, clk,
    output logic y
);
typedef enum {S0, S1, S2, S3, S4, S5} statetype;
statetype state, nextstate;
always_ff @(posedge clk)
begin
    state <= nextstate;
end

always_comb
begin
    case(state)
    S0:
    begin
        if (a)
            nextstate = S1;
        else
            nextstate = S0;
    end
    S1:
    begin
        if(a)
            nextstate = S1;
        else
            nextstate = S2;
    end
    S2:
    begin
        if(a)
            nextstate = S1;
        else
            nextstate = S3;
    end
    S3:
    begin
        if(a)
            nextstate = S4;
        else
            nextstate = S0;
    end
    S4:
    begin
        if(a)
            nextstate = S5;
        else
            nextstate = S2;
    end
    S5:
    begin
        if(a)
            nextstate = S1;
        else
            nextstate = S2;
    end
endcase

    case(state)
        S5:
        begin
          y = 1;
        end
    default:
         begin
          y = 0;
        end
endcase
end
endmodule
```

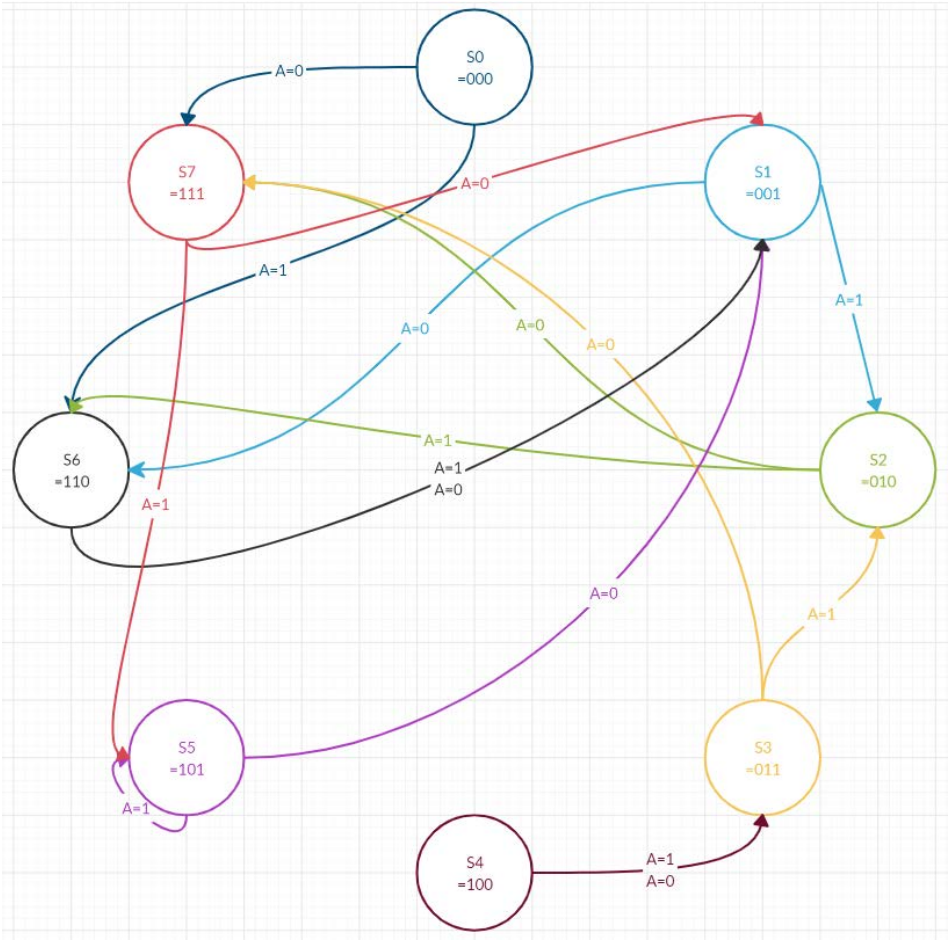


C. State Bulma

---

5. Şekil 3 de verilen devrenin state transition diagramını çıkarın. A da ki register'ı ihmal edebilirsiniz.

Öncelikle devrenin State'inin S0=000 olduğunu varsayarak, A=0 girişini yaparak incelemeye başlanıldı. Sırası ile devrenin State'lerini ve A'nın değerini değiştirerek devrenin yanıtını kayıt altına aldım. Bu yanıtlara göre FSM grafiği hazırladım.



D. Makina Dili

---

6. Şekil 4 te verilen ARM assembly kodunu makine diline çevirin. Komutları hexadecimal olarak yazın. Bu komutların başlangıç adresi 0x08000000 olarak alın. Her komut 32-bit olarak alın. Hafızada nasıl gözüktüğünü.

Assembly kodları, 32 bit, Little Endian ve Big Endian şeklinde gösterildi.

Assembly Code	32-Bit Value	Little Endian	Big Endian
// 0x080000000:	0000 1000 0000 0000 0000 0000 0000 0000	0x08000000	0x00000010
MOV R3, #137	0000 1000 0000 0000 0000 0000 1000 1001	0x08000089	0x91000010
MOV R2, #-1	0000 0111 1111 1111 1111 1111 1111 1111	0x07FFFFFF	0xFFFFFFE0
LSL R3, R3, #1	0001 0000 0000 0000 0000 0000 0000 0000	0x10000000	0x00000008
EOR R10, R2, R3	0001 0111 1111 1111 1111 1111 1111 1111	0x17FFFFFF	0xFFFFFFE8
STR R10, [R11], #4	1110 0100 1000 1011 1010 0000 0000 0100	0xE48BA004	0x2005D127

E. Makale Özetleri

---

7. D. Patterson, "50 Years of computer architecture: From the mainframe CPU to the domain-specific tpu and the open RISC-V instruction set," 2018 IEEE International Solid - State Circuits Conference - (ISSCC), San Francisco, CA, 2018, pp. 27-31, doi: 10.1109/ISSCC.2018.8310168.

IBM firması 1960 yıllarında, kendine özel olan bir mimariye, Instruction-Set Arcgitecture, sahipti. Bu temel mimari firmanın müşterilerine sunduğu bütün müşterilerine sunulmakta idi. 1974 Yılında IBM System/360 isimli ürün kataloğunu ortaya sundu. Bu sistem geçmişteki bilgisayar dizayn konseptlerinden fark yaratan bir sistemdir ve farklı versiyonları hala günümüzde, 50 yıl sonra, bile satılmaktadır.

Sektör bu değişiklerden sonra 1970'li yıllarda minicomputer'lara yöneldi. Bu bilgisayarların Ram'leri, Rom'ları ve Lojik elementlerinin hepsi aynı tür transistörlerden yapılmakta idi. Bu dönemde daha yüksek ISA'e sahip, daha gelişmiş microprogram'lar ortaya çıktı.

Intel'in kurucularından olan Gordon Moore, 1975 yılında Intel 8080'in tamamlanmasının ardından, Intel'in geleceğe yönelik kullanabileceği bir ISA için araştırmaları başlattı. Bu araştırmalar meyvelerini istenildiği kadar hızlı vermediğinde Intel hızlı bir şekilde, markette geride kalmamak için 8086 modelini acele ile piyasaya sürdü. Bu dönemlerde IBM, Apple I'e rakip olarak üretmek istediği

ürünü için Intel'in 8086 işlemcisi ile ilgilenmeye başladı. Bu ürünün beklenilenin çok üzerinde gerçekleşen başarısı, alelacele geliştirilmiş olan Intel 8086'yı büyük bir başarı haline getirdi.

1980'li yıllarda, sektörün mimariye yaklaşımı basitlik teması olmuştur. Geçtiğimiz yılların CISC mimarileri yerine, RISC mimarileri ortaya sundukları basitlik sayesinde daha küçük cache'leri ile bir çipe konulabilir hale geldi. Basitliklerinden ve hızlarından dolayı RISC'in o dönemde daha avantajlı olduğu düşünülmüştür.

Zamanlar 80x86 Mimarisi RISC mimarisini performansta geçmiştir, bundan dolayı günümüzde kişisel bilgisayarlarda ve sunucularda 80x86 mimarisi kullanılmaktadır. Buna rağmen RISC mimarisi ortadan kalkmamıştır, günümüzün küçük kişisel elektronik aletlerinde ve telefonlarında her zaman RISC mimarisine sahip elektronikler kullanılmaktadır.

VLIW bilgisayarların, RISC ve CISC'lerin yerini alacağı düşünülmekte idi. Fakat bu mimari donanım bazında tasarımları kolaylaştırmasına rağmen derleyicilerdeki yükü oldukça arttırmaktaydı.

1990'ların ortasında Intel, kullanımda olan 32 bit 80x86 mimarisinin yeterlilik döneminin sonuna gelindiğinin farkına vardı. Ayrıca Intel, market rakibi AMD ile ISA birlikteliğini de sonlandırmak istemekteydi. Bundan dolayı Intel, 1994 yılında HP ile bir mimari birlikteliğine girerek EPIC mimarisini öne çıkardı. AMD bu gelişmeden dışlatıldığı için kullanmakta olduğu mimariyi 64 bit'e uygun hale getirerek yetindi. Intel'in büyük umutlarına rağmen EPIC mimarisi, VLIW'dan dolayı, büyük bir başarısızlıkla sonuçlandı.

Sektörün bir sonraki trendi ise paralelleştirme oldu. Bir işlemcinin içine çok güçlü ve verimsiz büyük bir çekirdek koymak yerine, daha verimli birden fazla çekirdekten faydalanılmaya başlandı. Fakat bu paralelleştirme doğrusal olarak performans arttıramamaktadır, çekirdek sayısı arttıkça alınan verim azalmaktadır.

Normal işlemci tasarımlarına büyük bir değişiklik katan DSA mimarisinin en büyük örneği, Google'ın Tensor Processor'leri, sıradan CPU ve GPU'lara göre daha küçük bir çip alanında daha küçük güç tüketimleri ile 25 kat daha yüksek MAC'e sahiptir. Google bu konu ile ilgili çok bilgi paylaşmamasına rağmen sıradan sunuculara göre büyük avantajlara sahip olduğu söylenilmektedir.

DSA'lerin bu büyük başarılarına rağmen daha genel kullanımlara uygulanmaları mümkün olmamaktadır. Bundan dolayı market hala RISC işlemciler ile domine edilmektedir. Günümüzde daha güncel versiyonları olan ve açık kaynak olan RISC-V mimarisi bulunmaktadır. Bu minimalist ve modüler mimari günümüzde gelecek vadetmektedir.

Günümüzde hedef RISC-V mimarisini sunucular ve kişisel bilgisayarlar gibi elektronik aletlerde de kullanıma uygun bir hale getirmektir.

F. Referanslar

1. Harris and D. Harris, Digital Design and Computer Architecture: ARM Edition, 1st edition. Morgan Kaufmann, 2015
2. State Machine Diagram Tool, https://creately.com/lp/state-machine-diagram-tool/