Bu örneğimizde System Verilog diliyle temel bir ALU (Arithmetic Logic Unit) tasarımı yapacağız. Tasarıma geçmeden önce ALU'nun ne olduğu ve ne amaçla kullanıldığından bahsetmek gerekirse:

ALU NEDİR?

ALU (Arithmetic Logic Unit), bir işlemcinin aritmetik (toplama, çıkarma) ve mantıksal (AND, OR) işlemlerini gerçekleştiren temel bileşenidir. Tüm hesaplama kapasitesinin merkezinde yer alır ve işlemcinin çekirdeğini oluşturur.

MICROARCHITECTURE'DAKİ YERİ

Mikromimari, bir işlemcinin iç yapısını ve işlevlerini tanımlayan donanım organizasyonudur. ALU, burada "execute" (çalıştırma) aşamasının ana birimi olarak kullanılır ve kontrol biriminden gelen sinyallere göre matematiksel ve mantıksal işlemleri yürütür. Performans ve verimlilik, genellikle ALU'nun tasarımına ve optimizasyonuna bağlıdır.

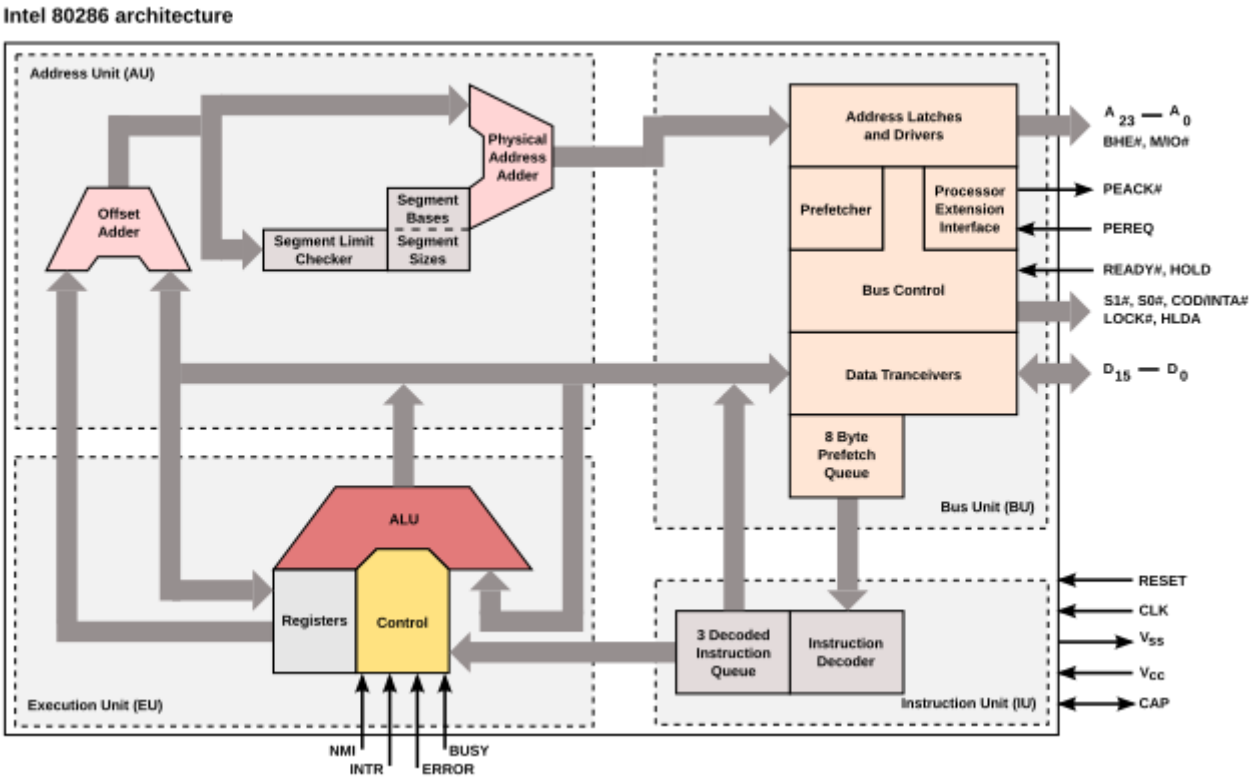


Figure 1: Intel 80286 microarchitecture

FPGA AÇISINDAN ÖNEMİ

FPGA'ler, mikroarchitecture düzeyinde özelleştirilmiş ALU'lar tasarlamak için idealdir. Tasarımcılar, belirli bir uygulama için ALU'yu optimize edebilir, paralel çalışan birden fazla ALU ekleyebilir veya pipeline yapılarıyla performansı artırabilir. Ayrıca, FPGA'ler özelleştirilmiş işlemci ve hızlandırıcı tasarımlarında ALU'nun işlevselliğini test etmek ve prototiplemek için esnek bir platform sağlar.

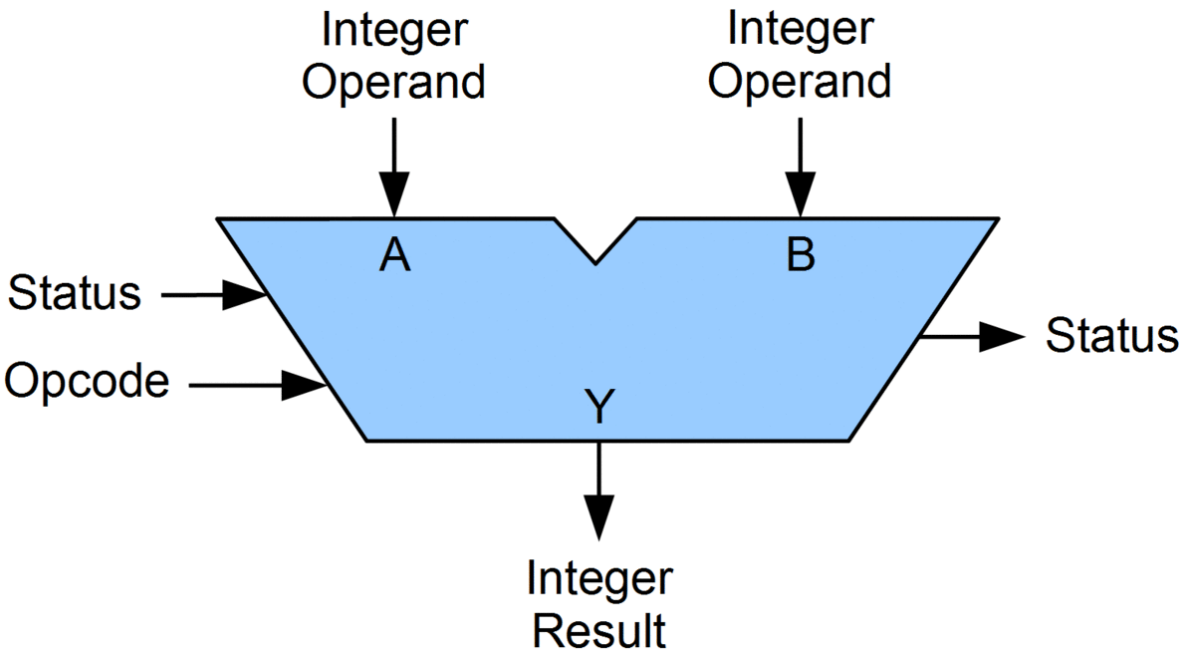


Figure 2: ALU

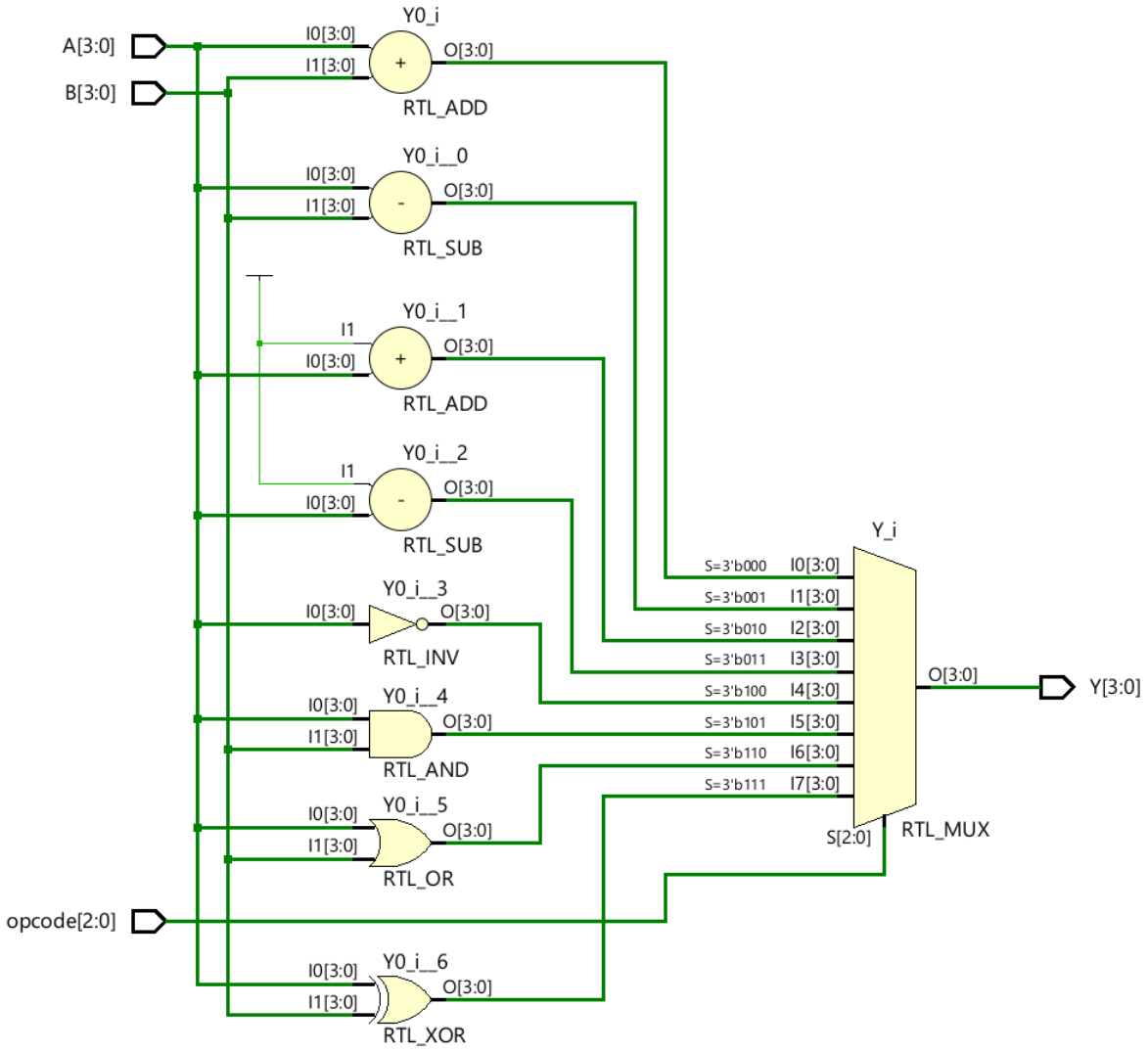
opcode	function	output
000	Add	A + B
001	Subtract	A – B
010	Decrrement	A – 1
011	Increment	A + 1
100	Invert	Not A
101	And	A and B
110	Or	A or B
111	Xor	A xor B

Design kodumuzu tasarlamaya başlayabiliriz:

```
module alu
#(
      parameter N = 4 //giriş-çıkışlarının kaç bitlik olacağını dinamik olarak
ayarlamayı sağlar
 )
 (
   input logic [N-1:0] A,
   input logic [N-1:0] B,
   input logic [2:0] opcode, //ALU'nun hangi işlemi gerçekleştireceğini
belirten bir kontrol sinyalidir.
   output logic [N-1:0] Y

 );

 always_comb
 begin
```

```
      case(opcode)
          0: Y = A + B; // toplama
          1: Y = A - B; // çıkarma
          2: Y = A + 1; // increase (artırma)
          3: Y = A - 1; // decrease (azaltma)
          4: Y = ~A; // tersi (invert)
          5: Y = A & B; // ve
          6: Y = A | B; // veya
          7: Y = A ^ B; // xor
          default: Y = '0; //case yapısında belirtilmeyen bir opcode değeri
geldiğinde devrenin nasıl davranacağını tanımlar.
      endcase
 end
endmodule
```

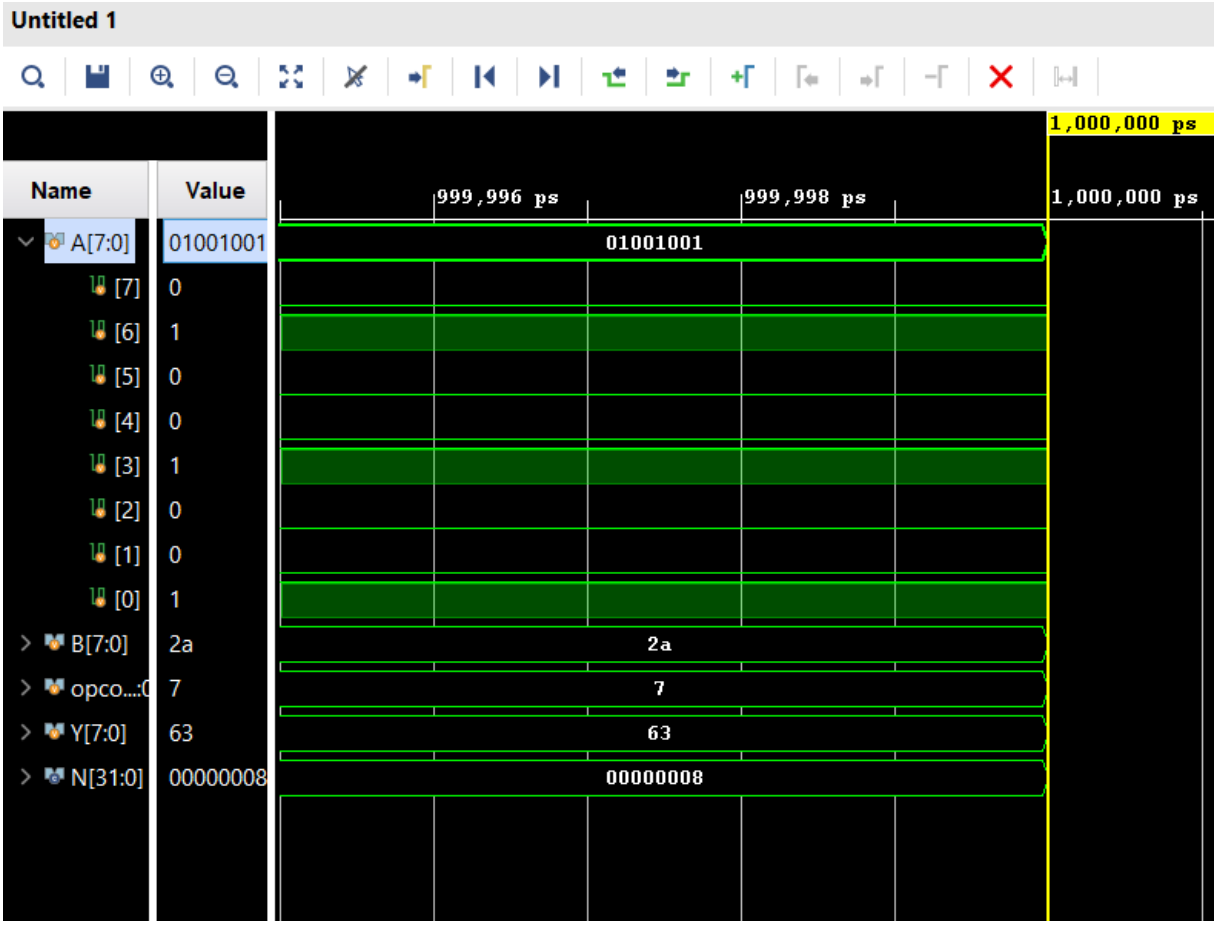
Design çıktımız şu şekilde olacaktır:



Şimdi de testbanch kodunu yazmaya başlayalım:

```
module tb_alu
#(
    parameter N = 8
);

logic [N-1:0] A;
logic [N-1:0] B;
logic [2:0] opcode;
logic [N-1:0] Y;

alu
#(
    .N(N)
)

alu_Inst
(
    .A(A),
    .B(B),
    .opcode(opcode),
    .Y(Y)
);

initial begin
    #5;
    A = 73;
    B = 42;

    for(int i = 0; i < 8; i++)
    begin
        opcode = i;
        #10;
    end
end
endmodule
```

Simülasyon çıktımız aşağıdaki gibi olacaktır:



Çıktıların değerini farklı sayı sistemlerinde görmek için istenen değere sağ tıklayıp > Radix > seçme işlemi yapabilirsiniz.