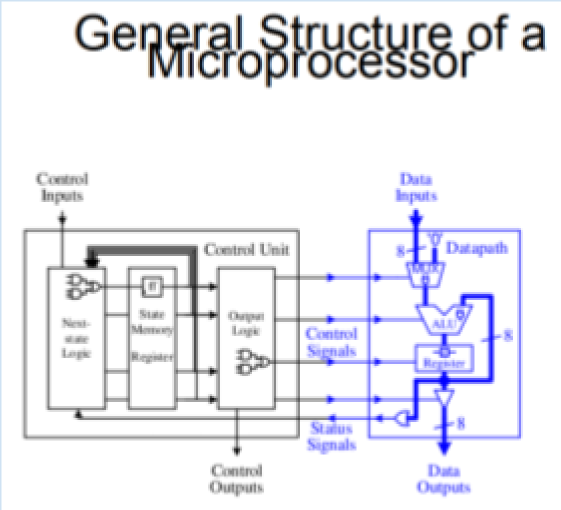
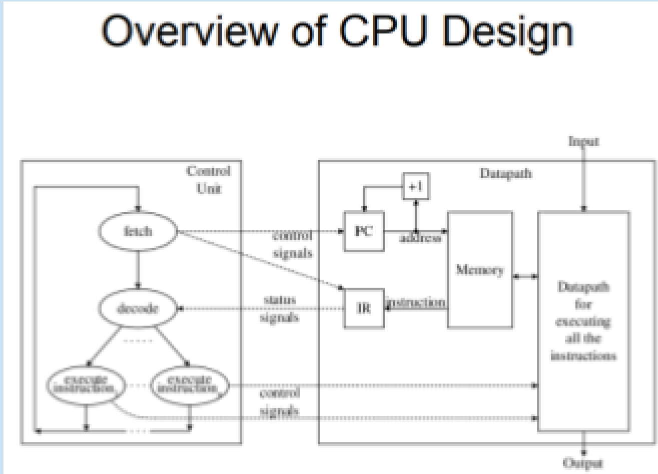


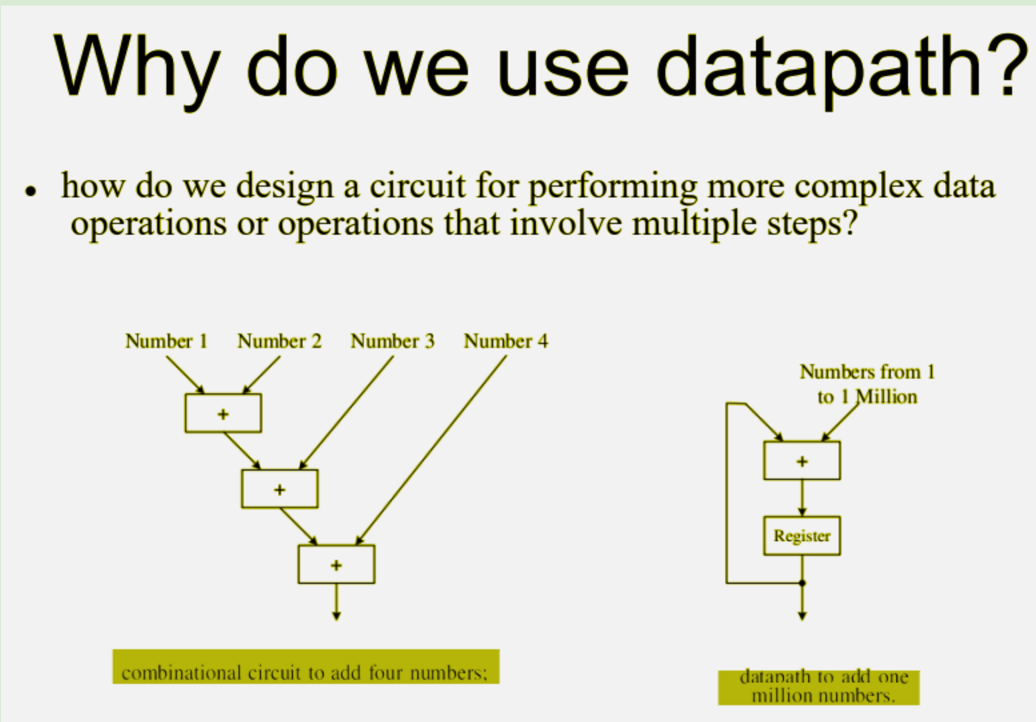


Datapath

Manipulates data. It includes:

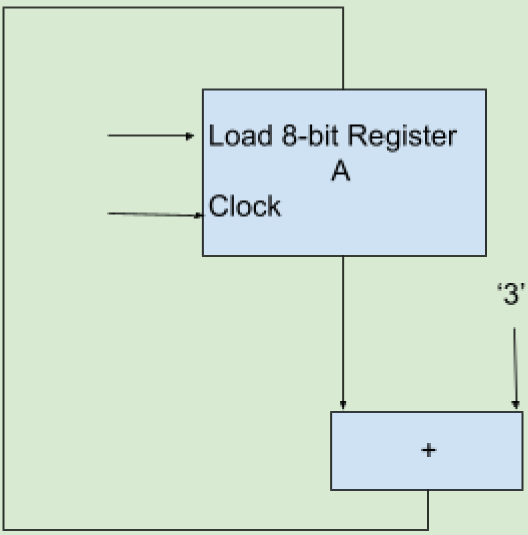
- Functional units: Adder, shifter, multiplier, ALU(Arithmetic Logic Unit), comparator
- Registers and other memory elements for the temporary storage of the data
- Buses, multiplexers and tri-state buffers for the transfer of data between the different components in datapath and the external world.

Control yoksa, sadece inputlar geliyorsa ona datapath demiyoruz.

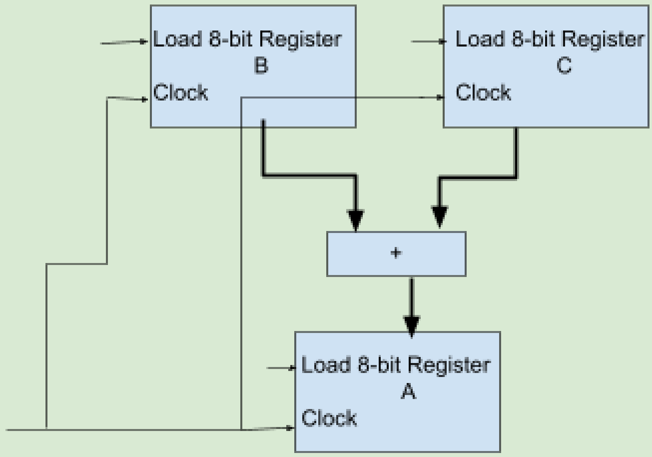


Designing Dedicated Datapaths:

A = A + 3



A = B + C



Multiplexers: Register'a göndereceğimiz **datayı seçmemizi sağlıyor,** min 2 inputlu oluyor 1 ve 0. Daha az adder Alu vs kullanmamızı sağlıyor.
Data Transfer: Multiple source olunca **multiplexer kullanarak data transfer etme olayı,** selection ile seçiyorsun hangi datayı transfer etmek istediğini.
Tri-State Bus: output gösterebilmek, bastırmak için kullanıyoruz.(**printf** gibi bişi aslında)

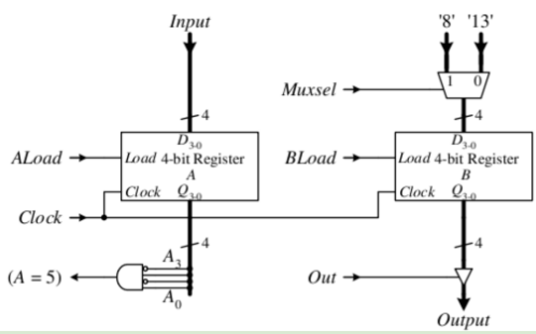
Generating Status Signals

Aslında olay condition(if statement gibi bir şey). Status signals are the results of the **conditional tests** that the datapath supplies to the control unit. *Caner anlatırken dersi or, and kapısı vs kullanmak yerine direkt adder "+" nasıl koyuyorsak "=" koyarak çizdi tahtaya. Sanırım sınavda da öyle yaparsak olur.*

Simple If-Then-Else

```
INPUT A
IF (A = 5) THEN
   B = 8
ELSE
   B = 13
END IF
OUTPUT B
```

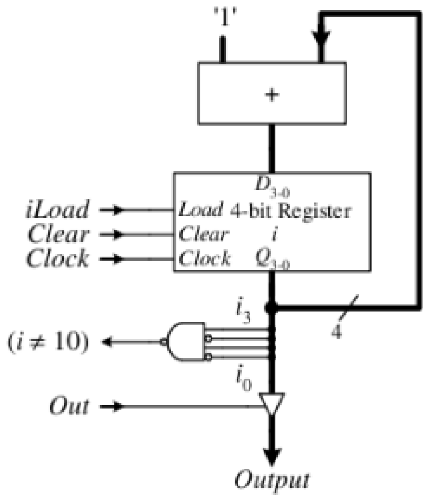
Control Word	Instruction	*ALoad*	*Muxsel*	*BLoad*	*Out*
1	INPUT *A*	1	×	0	0
2	*B* = 8	0	1	1	0
3	*B* = 13	0	0	1	0
4	OUTPUT *B*	0	×	0	1

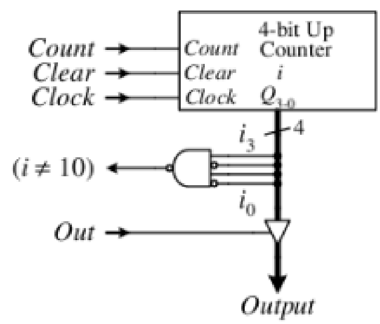


Example1

Counting 1 to 10

```
i = 0
WHILE (i ≠ 10){
    i = i + 1
    OUTPUT i
    }
```





Control Word	Instruction	*iLoad*	*Clear*	*Out*
1	$i = 0$	0	1	0
2	$i = i + 1$	1	0	0
3	OUTPUT i	0	0	1

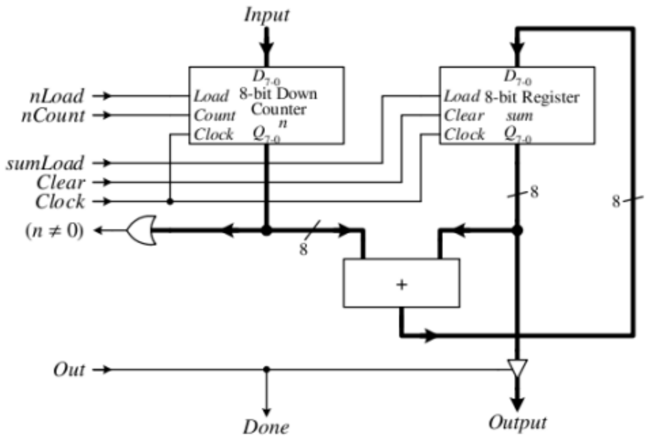
Control Word	Instruction	*Count*	*Clear*	*Out*
1	$i = 0$	0	1	0
2	$i = i + 1$	1	0	0
3	OUTPUT i	0	0	1

Tabloya while'ın içindeki i!=10 u eklemiyoruz çünkü bir statement değil, sadece checkliyoruz orda.
while adding not operator(i!=10 kısmı) to check whether it is equal or not, we dont need i selection column in the table cause we dont need a multiplexer.

Example2

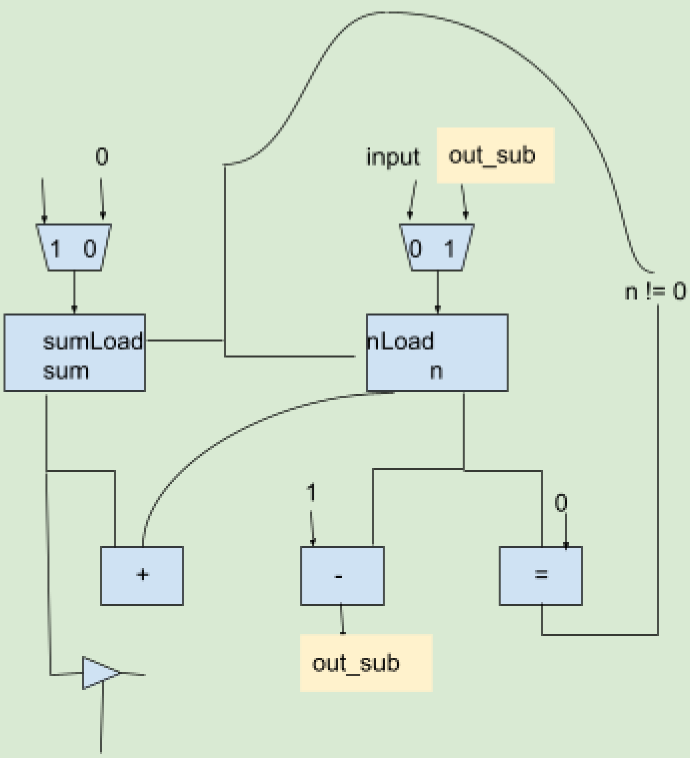
Summation of *n* Downto 1

```
sum = 0
INPUT n
WHILE (n ≠ 0){
    sum = sum + n
    n = n - 1
}
OUTPUT sum
```

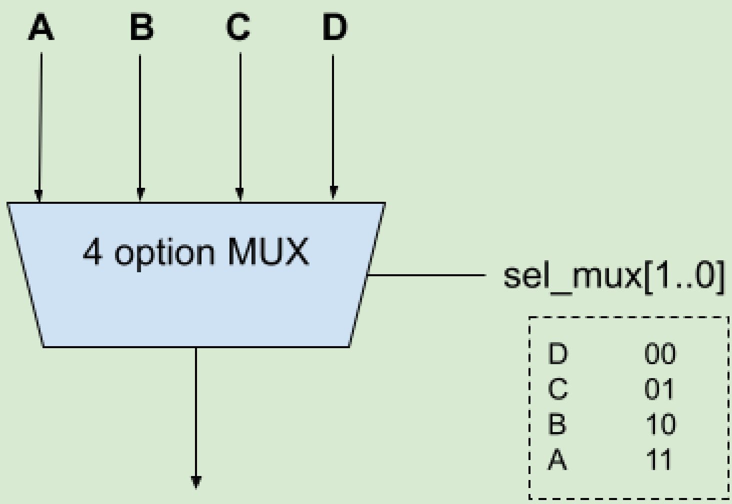


Control Word	Instruction	*nLoad*	*nCount*	*sumLoad*	*Clear*	*Out*
1	*sum* = 0	0	0	0	1	0
2	INPUT *n*	1	0	0	0	0
3	*sum* = *sum* + *n*	0	0	1	0	0
4	*n* = *n* – 1	0	1	0	0	0
5	OUTPUT *sum*	0	0	0	0	1

*loop olsun diye sumload ekliyoruz.**özellikle Caner'in yaptığında yani alttaki design** 8-bit "sum" register ındaki clear sayesinde multiplexer kullanıp sınıflamamız gerekmiyor ve tabloda column fazlalığından kurtulabiliriz.*



VHDL for Datapath
Components of ALU



Right shift ->
01001100
00100110

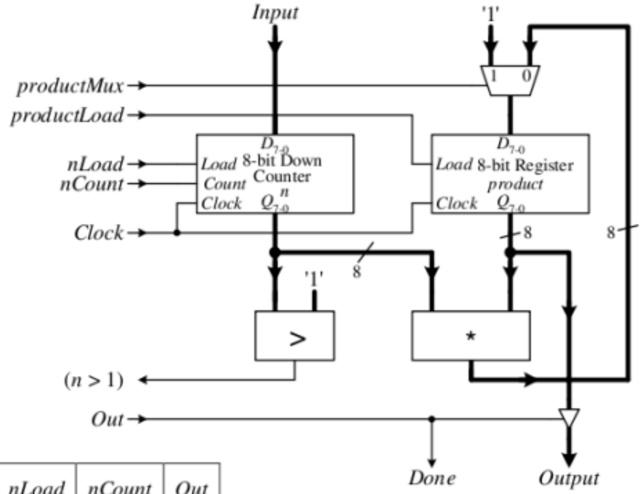
Rotate --)
01001001
10100100

Register File

A register file is a means of memory storage within a computer's **central processing unit** (**CPU**). The computer's register files contain bits of data and mapping locations. These locations specify certain addresses that are input components of a register file. Other inputs include data, a read and write function and execute function.

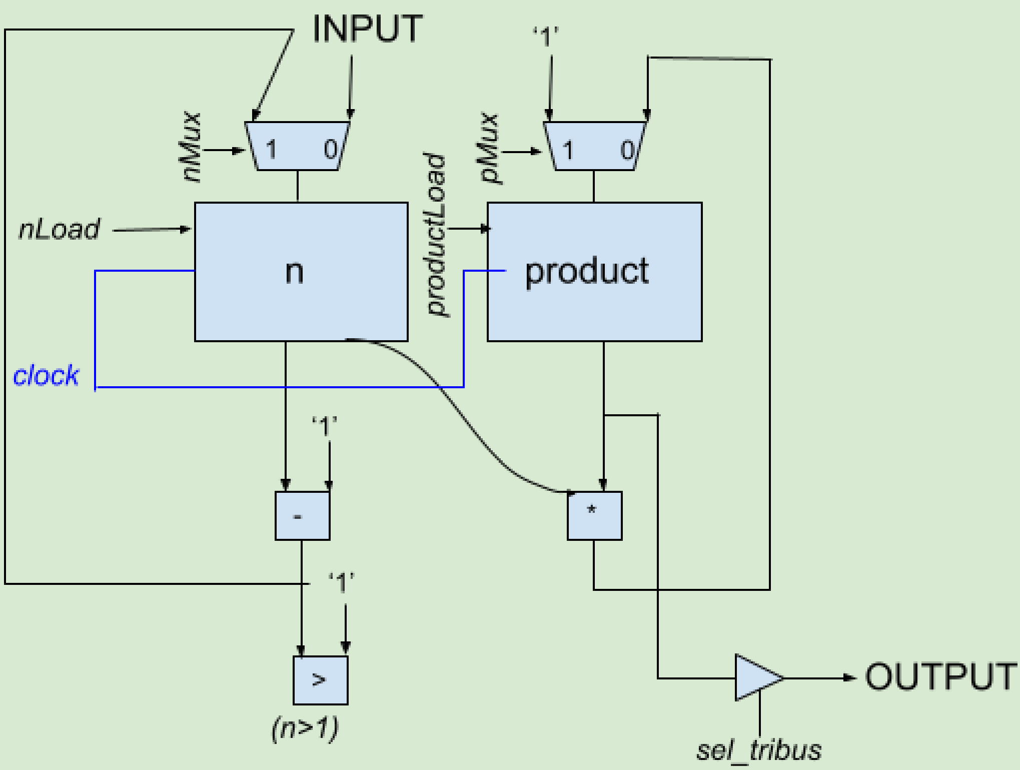
Factorial of *n*

```
1   INPUT n
2   product = 1
3   WHILE (n > 1) {
4       product = product * n
5       n = n - 1
    }
6   OUTPUT product
```



Control Word	Instruction	*productMux*	*productLoad*	*nLoad*	*nCount*	*Out*
1	INPUT *n*	×	0	1	0	0
2	*product* = 1	1	1	0	0	0
3	*product* = *product* * *n*	0	1	0	0	0
4	*n* = *n* – 1	×	0	0	1	0
5	OUTPUT *product*	×	0	0	0	1

Control Word	Instruction	*productMux*	*productLoad*	*nLoad*	*nCount*	*Out*
1	INPUT *n*, *product* = 1	1	1	1	0	0
2	*product* = *product* * *n*, *n* = *n* – 1	0	1	0	1	0
3	OUTPUT *product*	×	0	0	0	1

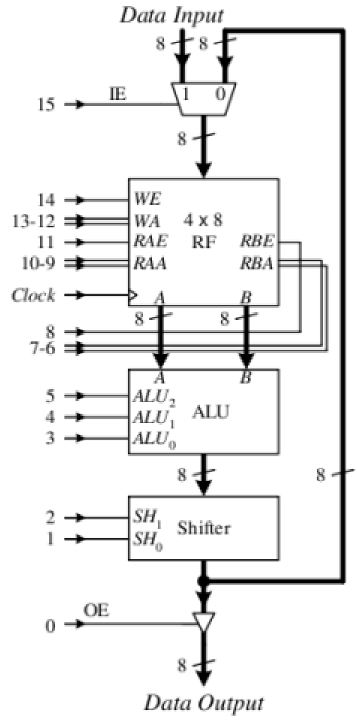


General Purpose Datapath

```
sum = 0
INPUT n
WHILE (n ≠ 0){
    sum = sum + n
    n = n - 1
}
OUTPUT sum
```

ALU_2	ALU_1	ALU_0	Operation
0	0	0	Pass through A
0	0	1	A AND B
0	1	0	A OR B
0	1	1	NOT A
1	0	0	$A + B$
1	0	1	$A - B$
1	1	0	$A + 1$
1	1	1	$A - 1$

SH_1	SH_0	Operation
0	0	Pass through
0	1	Shift left and fill with 0
1	0	Shift right and fill with 0
1	1	Rotate right



Control Word	Instruction	IE 15	WE 14	$WA_{1,0}$ 13–12	RAE 11	$RAA_{1,0}$ 10–9	RBE 8	$RBA_{1,0}$ 7–6	$ALU_{2,1,0}$ 5–3	$SH_{1,0}$ 2–1	OE 0
1	$sum = 0$	0	1	00	1	00	1	00	101 (subtract)	00	0
2	INPUT n	1	1	01	0	××	0	××	×××	××	0
3	$sum = sum + n$	0	1	00	1	00	1	01	100 (add)	00	0
4	$n = n - 1$	0	1	01	1	01	0	××	111 (decrement)	00	0
5	OUTPUT sum	×	0	××	1	00	0	××	000 (pass)	00	1

Example: Write the control words for manipulating the above circuit to perform the following program

Multiplication of two unsigned numbers:

```
prod = 0
INPUT A
INPUT B
WHILE (B ≠ 0){
   prod = prod + A
   B = B - 1
}
OUTPUT prod
```

Control Word	Instruction	IE 15	WE 14	$WA_{1,0}$ 13–12	RAE 11	$RAA_{1,0}$ 10–9	RBE 8	$RBA_{1,0}$ 7–6	$ALU_{2,1,0}$ 5–3	$SH_{1,0}$ 2–1	OE 0
1	*prod* = 0	0	1	00	1	00	1	00	101 (subtract)	00	0
2	INPUT *A*	1	1	01	0	××	0	××	×××	××	0
3	INPUT *B*	1	1	10	0	××	0	××	×××	××	0
4	*prod* = *prod* + *A*	0	1	00	1	00	1	01	100 (add)	00	0
5	*B* = *B* – 1	0	1	10	1	10	0	××	111 (decrement)	00	0
6	OUTPUT *prod*	×	0	××	1	00	0	××	000 (pass)	00	1