We need to design our CPU.

We have learnt all the MIPS instructions (arithmetic or logic) required by MIPS CPU. MIPS CPU is a machine that can execute only these instructions.

We first design ALU. It is a combinational circuit. It gives same output for same input. For complex operations like multiplication and division, sometimes sequential systems are used. Because in multiplication, you multiply the multipicant with each bit of the multiplier and then add the results. We can also design them as combinational circuits.

After we learn the ALU, we will use it as block diagram and we will design our datapath using ALU and all other components.

Unsigned Binary Integers

Typically to represent computer addresses that are guaranteed not to be negative.

n-bit unsigned binary integer (A = an-1an-2…a0) has a value of:

i

Range: 0 to 2n-1

Signed Binary Integers

n-bit 2’s complement binary integer (A = an-1an-2…a0) has a value of:

-an-1n-1 + i

Range: -2n-1 to 2n-1-1

2’s Complement Addition

Diagram

Description automatically generated with low confidence

Unsigned and 2’s complement addition are performed exactly the same way, but how they detect overflow differs.

2’s Complement Subtraction

To subtract 2’s complement numbers we first negate the 2nd number and then add the corresponding bits of both numbers.

Arrow

Description automatically generated with low confidence

Final carry olduğu için sonuç positive, olmasa negatifti ve 2’s complement formundaydı.

Sign magnitude system ---> 1000 0011 = -3 (en soldaki sign, kalanlar normal sayı)  
2’s complement ---> 1111 1101 = -3

Sign magnitude ile en büyük sayılar 🡪 27-1 = 127 , -127  
2’s complement ile en büyük sayılar 🡪 27-1 = 127 , -128

Why we prefer 2’s complement is about the subtraction. You cant perform subtraction using addition in sign magnitude system.

2’s complement of A = 2n- A = -A  
When I want to find B-A = B + (2n-A) = 2n + B – A 🡪 B-A is 32 bit number, 2n is n+1 bit number  
If you add these numbers, 2n will not change any bits in B-A

2n = 1000 0000 0000 … 🡪 n number of 0s and one 1. If you ignore 1 at the left, if you ignore the carry out you get from your adder, you are ignoring 2n.

For other systems, we need to design both adder and subtractor.

Bir sayının 2’s complementini almak sistem açısından çok kolay. 1 gatelik delayle yapılabilir.

Overflow

add 🡪 when there is an overflow, overflow flag will be raised and there will occur an exception.  
addu 🡪 even if there is an overflow, there will be no flag will be raised so exception will not occur

Table

Description automatically generated with medium confidence

Overflow occurs when you add 2 positive or 2 negative numbers.

Overflow doesn’t occur when you add 1 positive and 1 negative numbers

In addition,

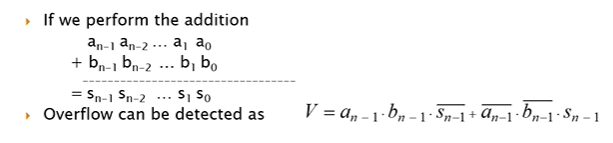
* if 2 operands are positive but result is negative 🡪 overflow
* if 2 operands are negative but result is positive 🡪 overflow

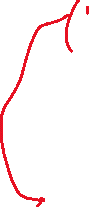
A + B = R → A3B3R3' + A3’B3’R3 = V → Then we have overflow (V is overflow bit)

2’s Complement – Detecting Overflow

When adding 2’s complement numbers, overflow will only occur if

* the numbers being added have the same sign
* the sign of the result is different





Overflow can also be detected as:

* V = cn cn-1 where cn-1  and cn are the carry in and carry out of the most significant bit. That means they are different.

Shift Operations

Text

Description automatically generated

Logical Operations

Table

Description automatically generated

ALU Interface

We will be designing a 32-bit ALU with the following interface:

Diagram

Description automatically generated

Words are 32-bit in MIPS so our ALU is 32-bit.

Set-on-less-than

slt $s1, $s2, $s3

This can be accomplished by

* subtracting $s3 from $s2
* setting the least significant bit to the sign bit of the result
* setting all other bits to 0
* if overflow occurs the sign bit needs to be inverted

Text, letter

Description automatically generated

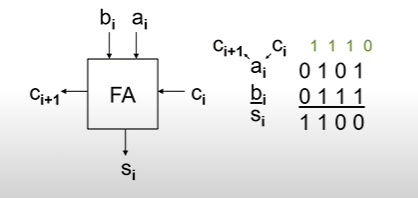
outputu götürüp s1’e yükleme görevi CPU’nun datapathinin ve control unitinin görevleri.

Full Adder

A fundamental building block in the ALU is a full adder (FA).

A FA performs a one bit addition.

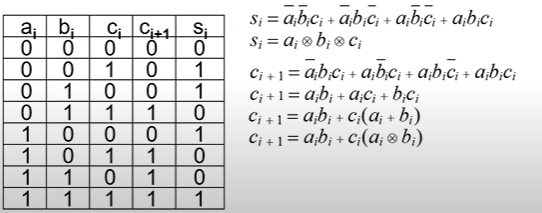
ai + bi + ci = 2ci+1 + si



**Full Adder Logic Equations**

si is ‘1’ if an odd number of inputs are ‘1’.

ci+1 is ‘1’ if two or more inputs are ‘1’.



**Full Adder Design (1-bit)**

Diagram

Description automatically generated

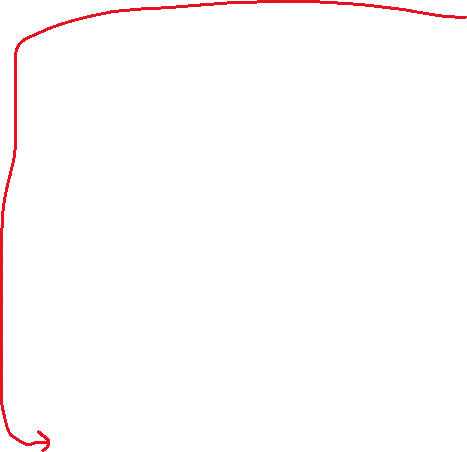
a’b + ab’ = ab

1-Bit ALU

The full adder, an xor gate, and a 4-to-1 mux are combined to form a 1-bit ALU.

Diagram

Description automatically generated



aibi  
ai + bi

Why we use this XOR? Isn’t it avoiding aibi at the 0?

* XOR is for subtraction. Only if the operation is subtraction or set-on-less-than, we have ALUop2 1.
* So XORing bi with 1 means that getting not bi. This XOR getting not bi when ALUop2 is 1, otherwise it gives bi
* Shortly, flip bi whenever there is a subtraction.
  + Flip second operand and do addition.

BITS IN ALUop: 0-0-0 🡪 ALUop2 – ALUop1 – ALUop0

* ALUop2 differs for operations that we need subtraction.

Addition and subtracting are in the same output (2 = 10)

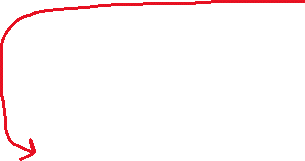
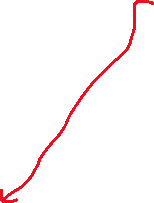
Output r will be 0 for all bits other than the least significant bit if set-on-less-than is selected.   
So, lessi will be connected to 0 for all bits other than the least significant bit.   
For least significant bit, lessi is connected to 1 or 0. It depends on if a is smaller than b or not.  
For now, forget about lessi

1-bit ALU for MSB

Diagram

Description automatically generated

We will compute the overflow and we will decide whether the result negative or not.



It is responsibility of CPU to raise a flag and result in exception or interrupt.

What is the purpose of this XOR gate?

* Overflow biti 0’sa zaten çıkarma sonucu MSB’i direkt dışarı veriyor.
* Overflow biti 1’se gelen sonucun complementini alır.
* Neden?
  + Overflow olunca taşmış oluyor. +’ysa – tarafa, -‘yse + tarafa geçmiş oluyor. Bunu geri çevirmemiz gerekiyor.
  + Overflow: 2 pozitifi topladık sonuç negatif çıktı VEYA 2 negatifi topladık sonuç pozitif çıktı.
  + Overflowun var olması demek, çıkan sonuç pozitifse gerçekte negatiftir, negatifse gerçekte pozitiftir.

Larger ALUs

Three 1-bit ALUs, a 1-bit MSB ALU, and a 4-input NOR gate can be concatenated to form a 4-bit ALU.

Diagram, schematic

Description automatically generated

We also want to know if the result is 0 or not. We need that for beq and bne instructions (subtract and if result is 0 they are equal).

Z is this 0 bit. It is 1 if result is 0.

Each ALU has 0 for set-on-less-than output because we know other than least significant bit, bits are 0 for set-on-less-than is selected.

If A<B, LSB must be “set” value.

Gate Counts

Diagram

Description automatically generated

1-bit ALU requires 16 gates.  
4-bit ALU requires 3x16 + (16+6) (for MSB ALU) = 70 + 3 + 0.5 (for 3 ORs and 1 inverter) = 73.5  
32-bit ALU requires 32x16 + 6 (2 XOR for MSB ALU) + (31x1) (OR gates in order to compute zero bit (n-1))  
 + 0.5 (1 inverter)

Gate Delays

Assume delays of

* 4-input mux = 2t
* XOR gate = 2t
* AND/OR gate = 1t
* Inverter = 1t

Additional delay needed to compute Z

We use term: propagation delay

What is the delay of?

* 1-bit ALU
  + We cant sum up all the delays because gates can work in parallel. One may not be have to wait for other.
  + Assume delays are constant and gates are vertices of a graph.
  + There is a linear time algorithm: static timing analysis 🡪 we find the worst delay
  + We visit each vertex only once. That’s why algorithm is linear.
  + Longest path must be found. In order to find it, first of all, circuit (or graph structure) topologically sorted which means that all gates which have an output that is connected to another gate must be visited before. After, we start from the first gate.
  + Giren inputlardan en geç gelen ile gate’in delayını toplayıp çıkan wire’a yazıyorsun.
  + Her gelen input (bi, ai, ALUop2,…) ilk başta 0 sayılır.

Diagram, schematic

Description automatically generatedHere red path is critical path of this circuit.

En alttan başla, en çok sürenleri takip et.

Optimize etmek istiyorsan bu yolu optimize etmelisin. Sonra başka critical path çıkabilir, onu da optimize et. Delay requirements ile satisfy olana kadar devam et.

Diagram, schematic

Description automatically generated

Here r0 will be at 10t (you can calculate), r3 will be latest because it needs to wait for c3. c3 is dependent on c2, c2 is dependent on c1. c1 is dependent on c0.

It is good to know the time difference between carry in and carry out. We get 7t time difference for the first ALU. c1 will come at 7t for the second ALU.

Diagram, schematic

Description automatically generatedBetween carry in and output, we have 5t delay.

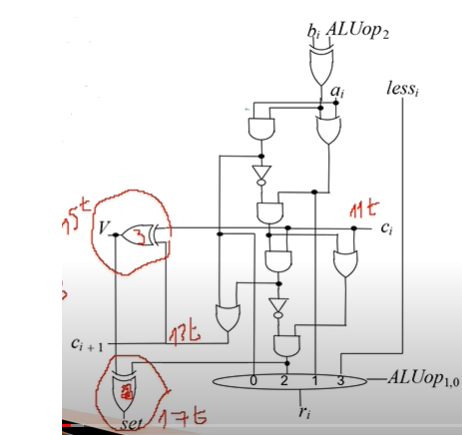


Between carry in and carry out, we have 2t delay.

So c2 will be ready at 9t, r1 will be ready at 12t.

c3 will be ready at 11t (9+2), r2 will be ready at 14t (9+5).

c4 will be ready at 13t, r3 will be ready at 16t.

V will be ready at 15t, set will be ready at 17t.

So set is connected to first ALU. Therefore, r0 will be ready at 19t (17+2 for mux).

This 19t is for set-on-less-than. Other instructions don’t use this set.

For other operations:

Diagram, schematic

Description automatically generated



Actual worst case delay of all circuit is 19t. This is (Z) useful only for beq or bne. For set-on-less-than we have 19t delay (r0). For all other operations we have 16t delay (r3).

Ripple Carry Adder (RCA)

With the previous design the carry “rippled” from one 1-bit ALU to the next

Diagram, schematic

Description automatically generated

This leads to a relatively slow design

* Az yer kaplar ama delay olarak kötü

Z is ready at 19t

Gate Delays

An n-bit ALU? 🡪 (2n + 8)t 🡪 other than beq,bne, set-on-less-than

* 7t (c1) + 2tx31 (her carry arası 2 bit) = 69t (c32) + 3 (c4 ile r3 arası fark) = 72t 🡪 delay for 32 bit ALU if we ignore Z bit.
* In order to OR 32 bit by using 2 input ORs, we need 5 levels of OR gates:
  + 16 🡪 1st level
  + 8 🡪 2nd level
  + 4 🡪 3rd level
  + 2 🡪 4th level
  + 1 🡪 5th level
  + Each level put additional 1t delay. Total 5t.
* 72t + 5t + 1t (for inverter) = 78t
* levels of 2-input OR gates and 1 inverter are needed to compute Z.

Carry Lookahead Adder (CLA)

With a CLA, the carries are computed in parallel using carry lookahead logic (CLL).

Diagram, schematic

Description automatically generated

Her bitten 2 bilgi alıyoruz: p (propagate) ve g (generate)

O bilgileri parallel bir şekilde işleyerek her basamağın bir öncekini beklemesine engel olmaya çalışıyoruz.

Her basamaktan bilgiler alarak c3’ün 1 mi 0 mı olacağını, c2’nin 1 mi 0 mı olacağını bulacağız.

Generate sinyalinin 1 olması ----> o bit carry’yi kendisi üretiyor  
Propagate sinyalinin 1 olması ---> o bit carry üretemiyor ama kendisine gelen carry’yi bir sonraki basamağa aktarabiliyor

4 case var:

* ai = 0, bi = 0
  + There will be no carry out whatever the carry in is.
* ai = 0, bi = 1 && ai = 1, bi = 0 (pi : propagate situation for i bit)
  + If carry in is 1, there will be carry out.
  + If carry in is 0, there will be no carry out.
  + pi will be 1 whenever the bit can not generate but it can propagate the carry.
    - pi = aibi
    - Also we can say: pi = ai + bi
      * because when generate is 1, propagate is also 1.
* ai = 1, bi = 1 (gi : generate situation for i bit)
  + carry out will be generated, you don’t have to wait for carry in.
  + gi will be 1 whenever the bit can generate a carry.
    - gi = ai.bi

Carry Logic Equation

The carry logic equation is

* ci+1 = aibi + (ai + bi)ci
* ci+1 = gi+ pici

Carry Lookahead Logic

For a 4-bit carry lookahead adder, the carries are computed as

Text

Description automatically generated

c leri birbirirlerinden bağımsız hale getirdik.

c0 is ready at t = 0.

We can compute gi and pi in parallel. For example g2 doesn’t need g1 and g0.

If each logic level has a delay of only 1t, the CLL has a delay of 2t 🡪 In practice this may not be realistic.

All these pi and gi signals are ready at same time = 3t.

4-bit CLA Timing

With a carry lookahead adder, the carries are computed in parallel using carry lookahead logic.

Diagram, schematic

Description automatically generated

This design requires 15x4 (16’dan 15’e düştük çünkü artık carry outu hesaplamıyoruz) + 14 = 74 gates without computing V or Z. Eskiden 16x4 = 64’tü yani gate sayısı 10 arttı.  
carry in ile output arası 5t, c3 5t ise r3 10t olur.

Modifying the 1-bit ALU

Diagram

Description automatically generated



We want gi and pi be correct even for subtraction so we don’t directly use 1 and gate for gi and 1 or gate for pi.

Text, letter

Description automatically generated

1 OR + 4 AND

1 OR + 3 AND

1 OR + 2 AND

1 OR + 1 AND

We need 14 additional gates in Carry Lookahead Logic.

p0 is ready at 3t. p0c0 is ready at 4t. g0 is ready at 3t. g0+p0c0 is ready at 5t.

p1 and g0 both 3t, p1g0: 4t. p1p0c0: 4t. c2: 5t

p2g1: 4t, p2p1g0: 4t, p2p1p0c0: 4t. c3: 5t

p3g2: 4t, p3p2g1: 4t, p3p2p1g0: 4t, p3p2p1p0c0: 4t, c4: 5t

Previously c1: 7t, c2: 9t, c3: 11t, c4: 13t. Now we have 5t for all.

16-bit ALU – VERSION 1

A 16-bit ALU could be constructed by concatenating four 4-bit CLAs and letting the carry “ripple” between 4-bit “blocks”.

Diagram, schematic

Description automatically generated

This design requires 74x4 = 296 gates, without computing V or Z.

c4 5t olduğu için buradaki c0lar c4 olacak, c4 olunca AND işlemleri 6t zaman alacak. Bir de OR var yani c8 7t zaman alacak.

c12 için de c0 yerine c8 kullanılacak.

16-bit ALU – VERSION 2

Another approach is to use a second level of carry lookahead logic.

This approach is faster, but requires more gates: 16x15 + 5x14 = 310 gates

Diagram

Description automatically generated

P ve G lerin hepsi 1 bitlik sinyaller. P3:0 demek 0 biti ile 3 biti arasındaki 4 basamağın propagate sinyali.

a3a2a1a0 g3g2g1g0  
b3b2b1b0 p3p2p1p0

* g3 1’se generate edilecek. MSB carry generate ediyorsa demekki 4 basamaktan dışarıya carry çıkacak.
* 2. basamak carry üretiyor, 3. basamak da bunu dışarı aktarıyorsa carry çıkar.
* 1. basamak carry üretiyor, 3. ve 2. basamaklar da propagate ediyor. Yani 1’de üretilen 2 ve 3 tarafından propagate edilecek ve carry dışarı çıkacak.
* 0. basamak carry üretiyor, diğerleri propagate ederek carrynin dışarı çıkmasını sağlıyor.

G3:0 = g3 + p3g2 + p3p2g1 + p3p2p1g0

P3:0 = p3p2p1p0 🡪 gelen carry tek tek dışarı aktarılır.

4-bit CLA

The 4-bit CLA (block CLA) is similar to the first 4-bit CLA, except the CLL computes a “block” generate and “block propagate”, instead of a carry out.

Thus the computation:

c4 = g3 + p3g2 + p3p2g1 + p3p2g0 + p3p2p0c0

is replaced by

G3:0 = g3 + p3g2 + p3p2g1 + p3p2p1g0  
P3:0 = p3p2p1p0

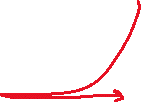
Note: c4 = G3:0 + P3:0 c0

This approach limits the maximum fan-in to four, and the carry-lookahead logic still requires 14 gates.

Şöyle bir değişim olacak:

A picture containing text

Description automatically generated



bunların da isimleri değişecek

P’ler 4t sürede, G’ler 5t sürede hazır olur.

Böylece version1’de 11t’de hazır olan c16, artık 7t’de hazır olacak.

Sadece c4 6t, diğerleri 7t.