İşaretli 2'li Sayılar (Signed Binary Numbers)

10'luk sistemde ve matematikte işaretli (eksi) sayılar başlarında "-" sembolü ile ifade edilirler:

-9, -12, -3.14

2'lik sistemde en anlamlı bit (MSB) 1 ise sayı eksi, 0 ise artı işaretlidir.
2'lik sistemde işaretli sayıları göstermek için meşhur olan 3 farklı yöntem mevucttur:

- İşaretli Büyüklük (Signed Magnitude)
- İşaretli 1'e Tümleyen (Signed 1's Complement)
- İşaretli 2'ye Tümleyen (Signed 2's Complement)

2'lik sistemde bir bit dizisi verildiğinde ilk olarak sayının hangi metotla ifade edildiğinin belirtilmesi gerekir:

01001: 9
11001: ???
Eğer bu dizi işaretsiz (unsigned) olarak tanımlandıysa 25 sayısını ifade eder
Peki işaretli (signed) olarak tanımlandıysa hangi sayıyı ifade eder?

İşaretli 2'li Sayılar (Signed Binary Numbers)

11001 = ???

- İşaretli Büyüklük: En anlamlı bit 1 ise, geri kalan bit dizisi işaretsiz olarak değeri hesaplanır ve eksi işaret eklenir:
11001: $-(1001)_2$ → -9

- İşaretli 1'e Tümleyen: En anlamlı bit 1 ise, geri kalan bit dizisinin 1'e tümleyeni alınır ve eksi işaret eklenir:
11001: $-(0110)_2$ → -6

- İşaretli 2'ye Tümleyen: En anlamlı bit 1 ise, geri kalan bit dizisinin 2'ye tümleyeni alınır ve eksi işaret eklenir:
11001: $-(0111)_2$ → -7

NOT: Günümüz bilgisayarlarda İşaretli 2'ye Tümleyen (Signed 2's Complement) metodu kullanılmaktadır

İşaretli 2'ye Tümleyen'de eksi sayıları hesaplama:

11001: -1*2^4 + 1*2^3 + 1 * 2^0 = -7

İşaretli 2'li Sayılar (Signed Binary Numbers)

Günümüz bilgisayarları neden 2'ye tümleyen metodunu kullanıyorlar?

5 bit ile İşaretli Büyüklük yöntemiyle gösterilebilecek sayılar: -15,-14,…,-1,-0,+0,1,…,15
5 bit ile İşaretli 1'e Tümleyen yöntemiyle gösterilebilecek sayılar: -15,-14,…,-1,-0,+0,1,…,15
5 bit ile İşaretli 2'ye Tümleyen yöntemiyle gösterilebilecek sayılar: -16,-15,…,-1,0,1,…,15

Toplama/Çıkarma İşlemleri:

+ 6	00000110	− 6	11111010
+13	00001101	+13	00001101
+19	00010011	+ 7	00000111
+ 6	00000110	− 6	11111010
−13	11110011	−13	11110011
− 7	11111001	−19	11101101

Taşma (Overflow)

Bilgisayarlarda (ya da 2'li aritmetik yapan herhangi bir sayısal devrede) aritmetik işlemlerin yapıldığı birimlerin sınırlı alana sahip olduğunu hiçbir zaman unutmamak gerekir.

Örnek olarak 4-bit büyüklüğünde sayılarla işlem yaptığımızı düşünelim → -8,-7,…,6,7

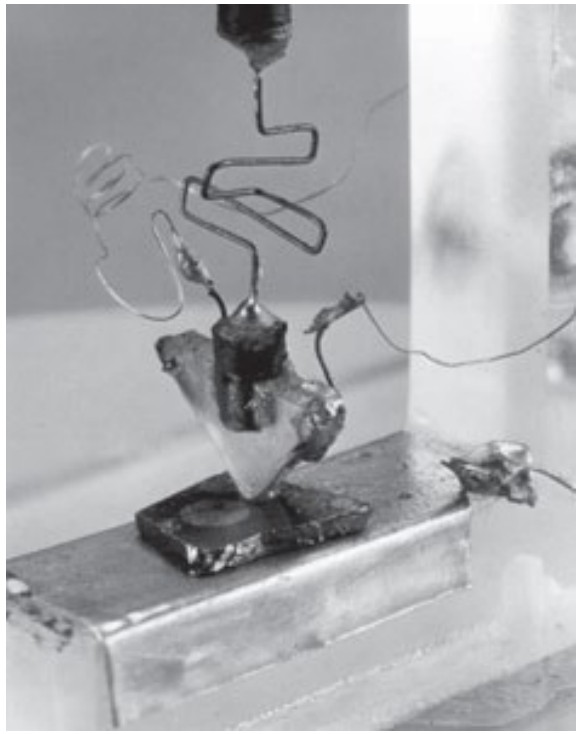
1001: - 7	0111: +7	0111: +7	1101: -3
+ 0111: +7	− 1011: -5	+ 0101: +5	+ 1010: -6
10000: 0	????: +12	1100: -4	10111: +7

Bilgisayarlarda (ve pek çok sayısal sistemde) kullanılan standart veri tipleri (C dili baz alınmıştır):

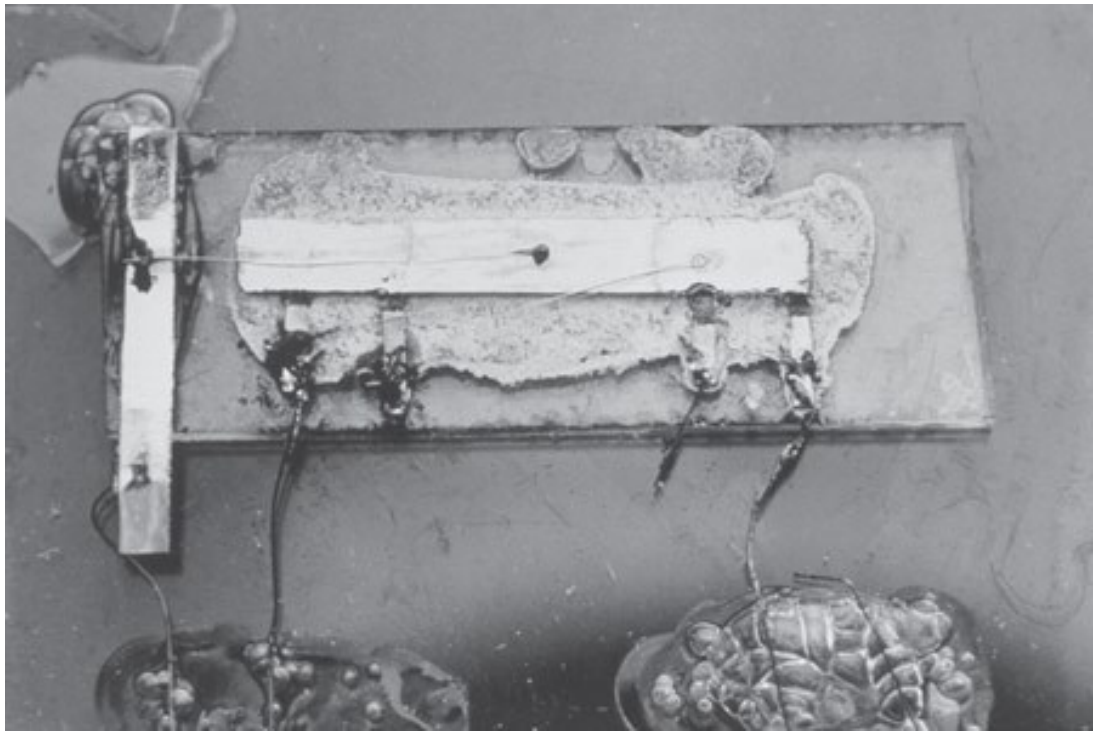
unsigned char	: 8-bit işaretsiz sayılar	→ 0,1,…,255
signed char	: 8-bit işaretli sayılar	→ -128,-127,…,126,127 (-2^7,…,2^7-1)
int	: 32-bit işaretli sayılar	→ -2_147_483_648,…, 2_147_483_647 (-2^31,…,2^31-1)
long	: 64-bit işaretli sayılar	→ (-2^63,…,2^63-1)
float	: 32-bit kayan sayılar	→ (-1.2E-38 to 3.4E+38)
double	: 64-bit kayan sayılar	→ (-2.3E-308 to 1.7E+308)

İlk Transistör – İlk Tümleşik Devre

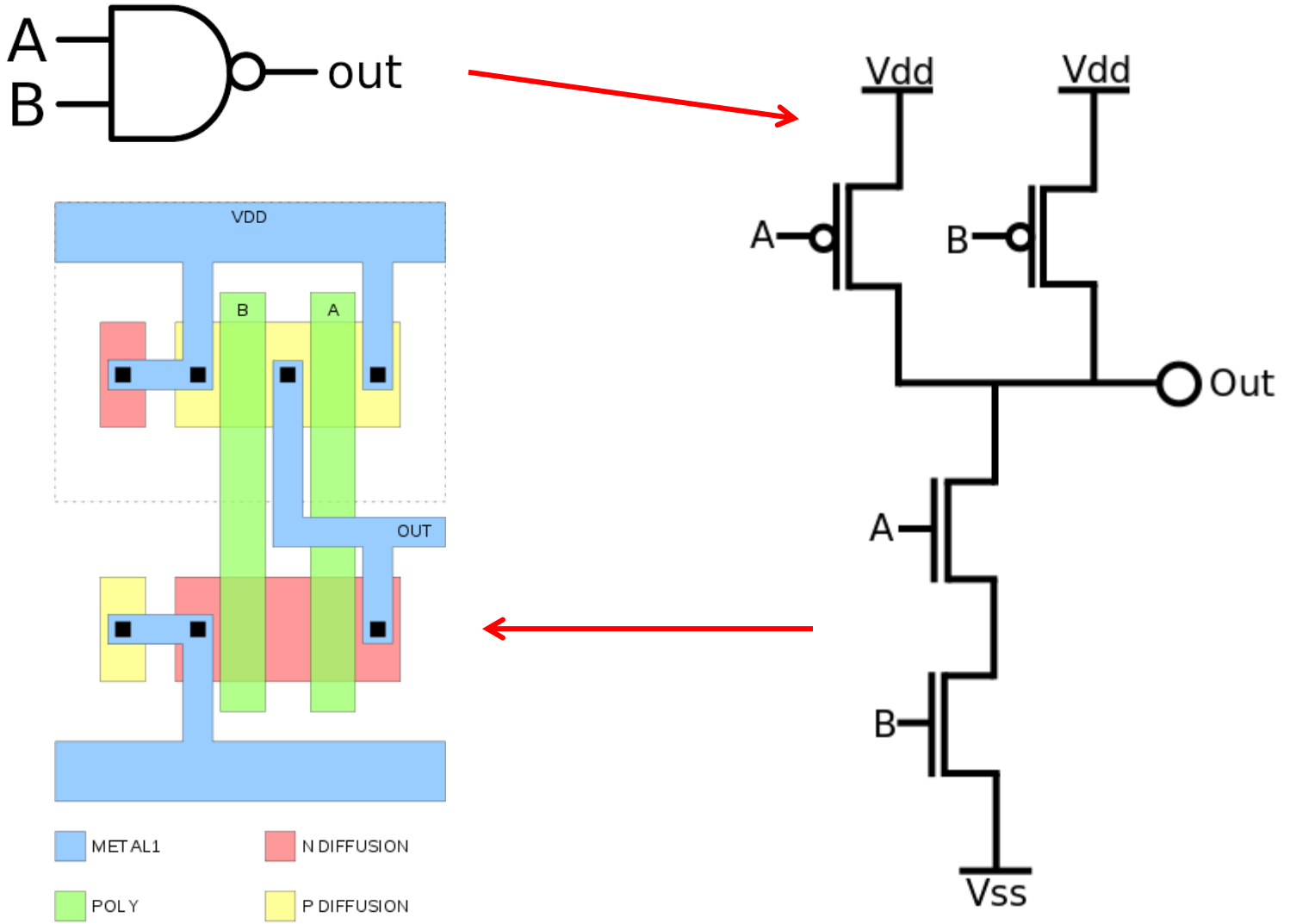
1947 – john bardeen, walter brattain, william shockley
İlk transistor – 1956 nobel fizik ödülü



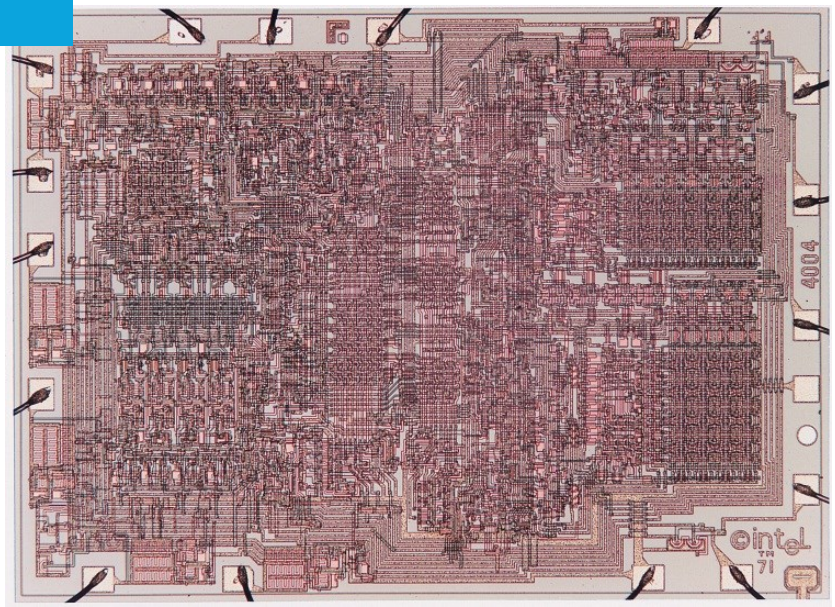
1958 – jack kilby İlk Tümleşik Devre – Flip Flop
2000 nobel prize for physic

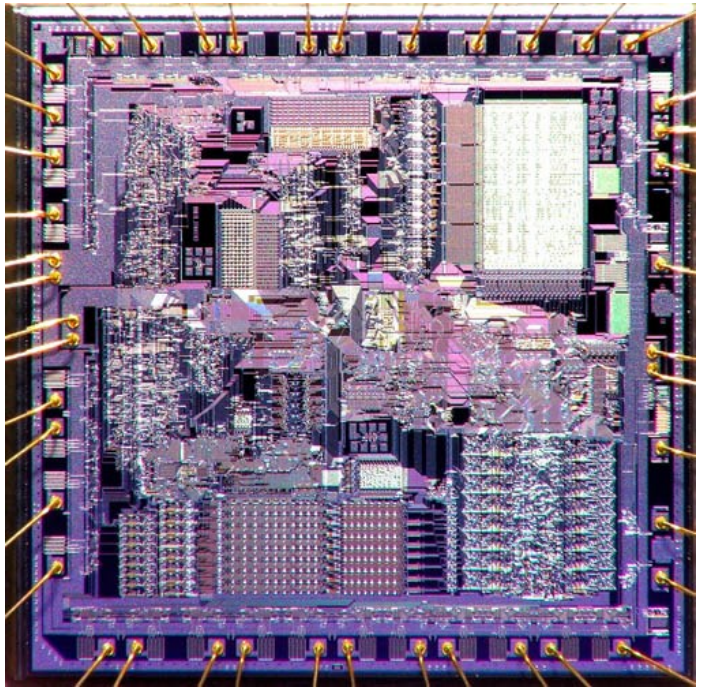


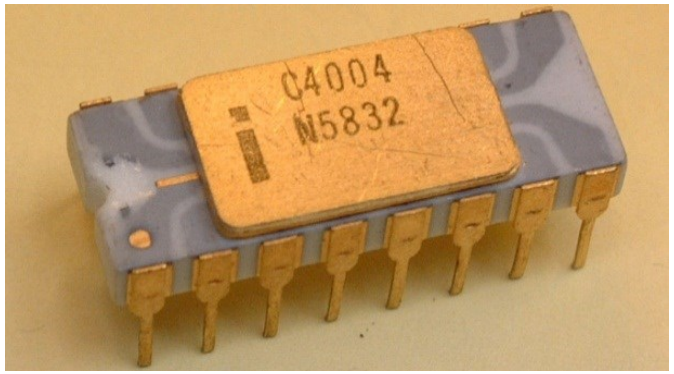
Kapı (Gate) – Transistör – Yerleşim (Layout)

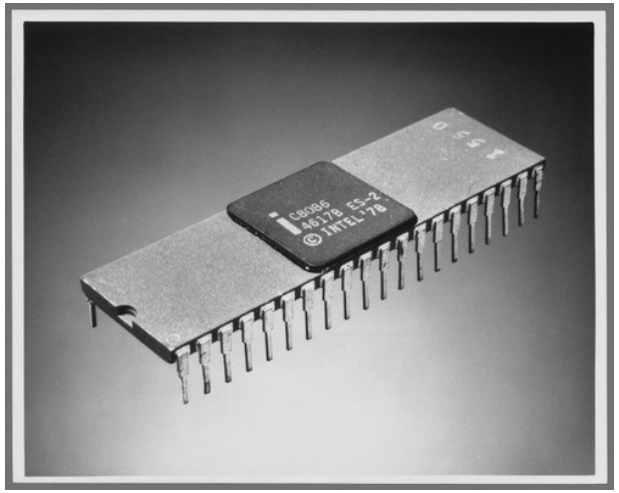


Mikroişlemciler

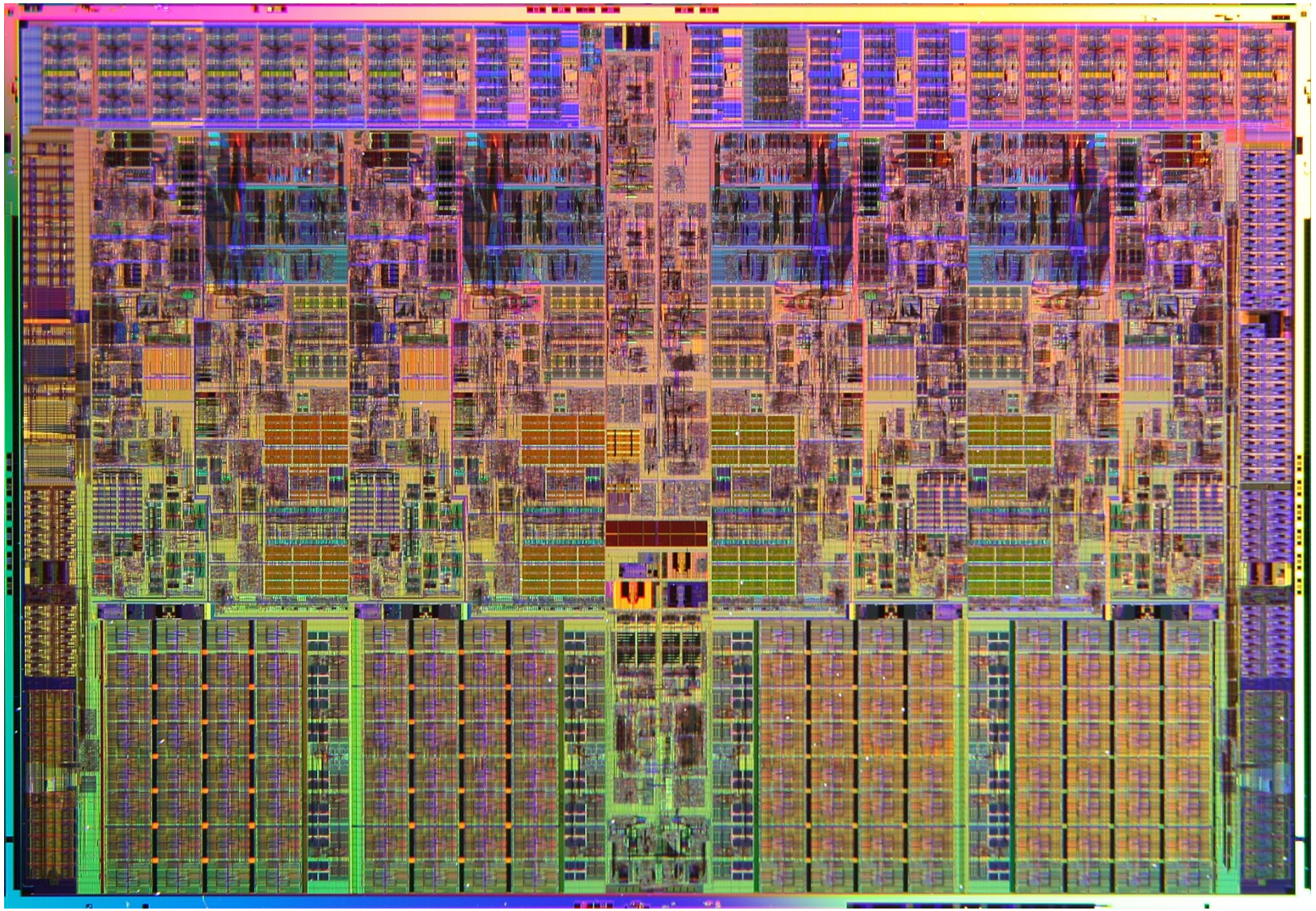






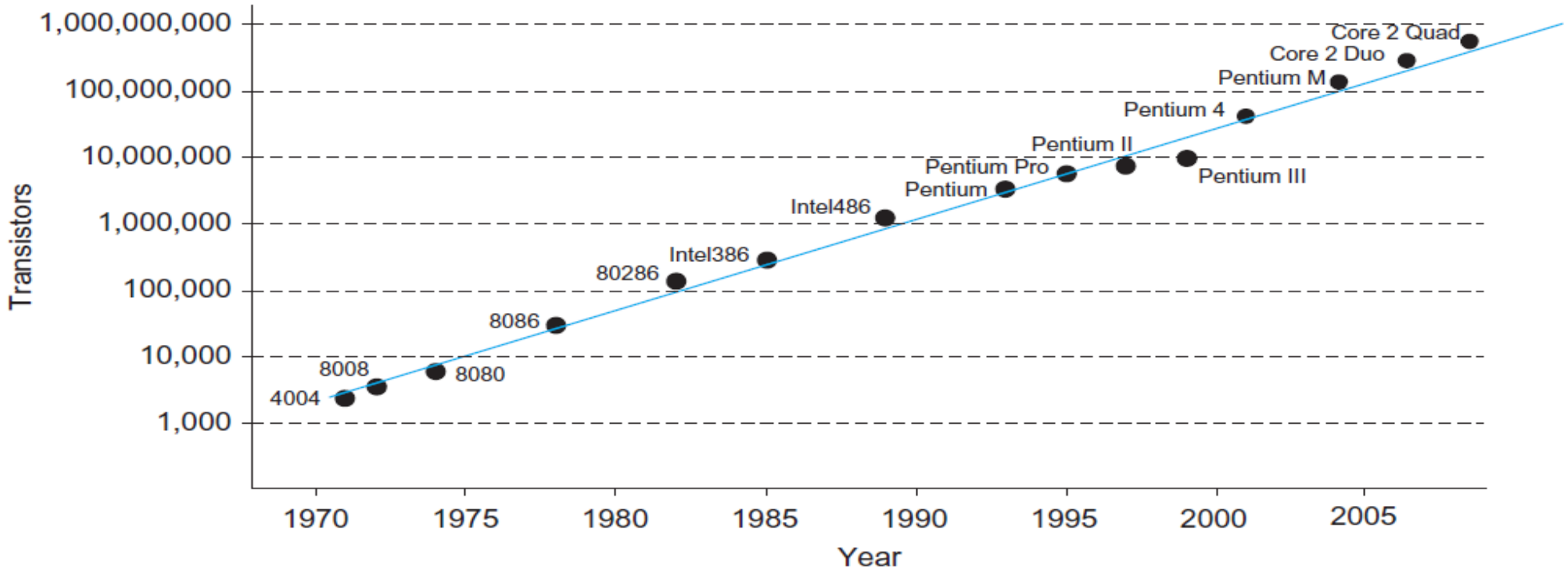


Mikroişlemciler

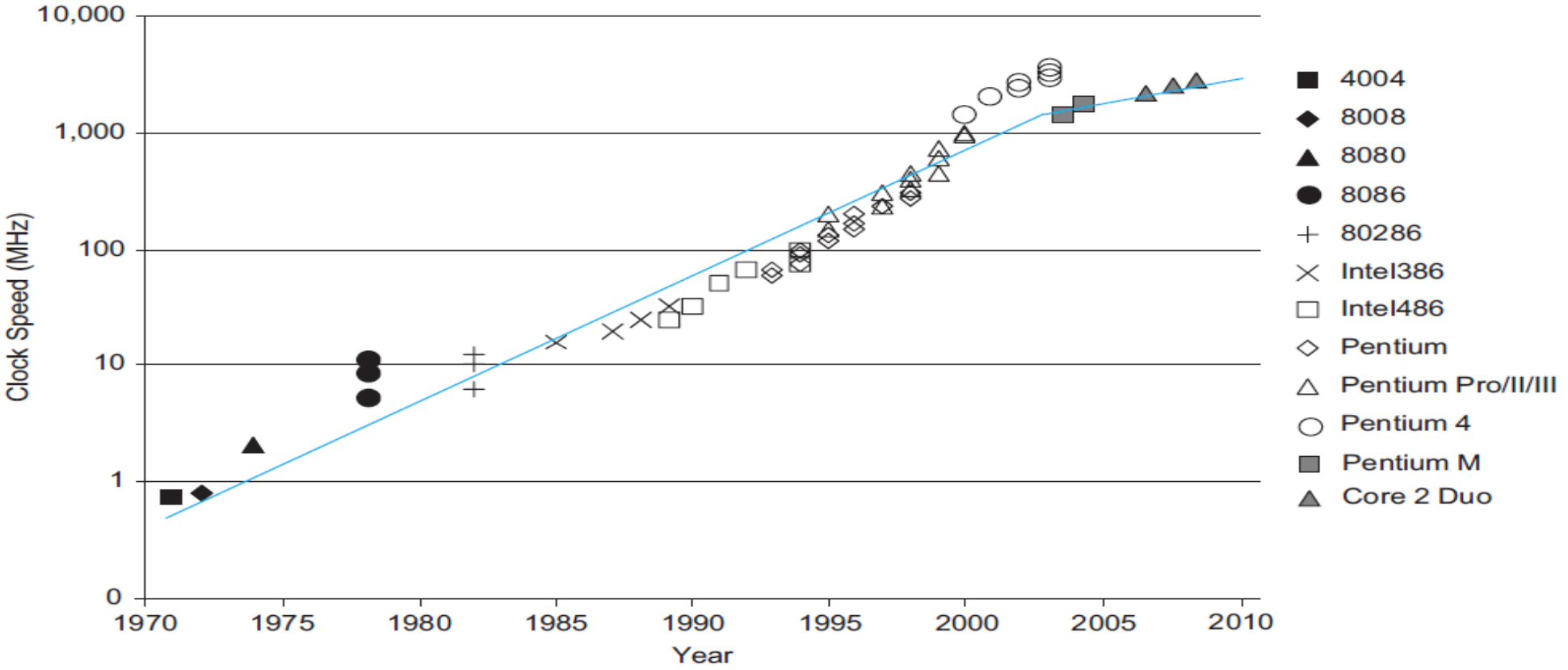


MOORE'S LAW

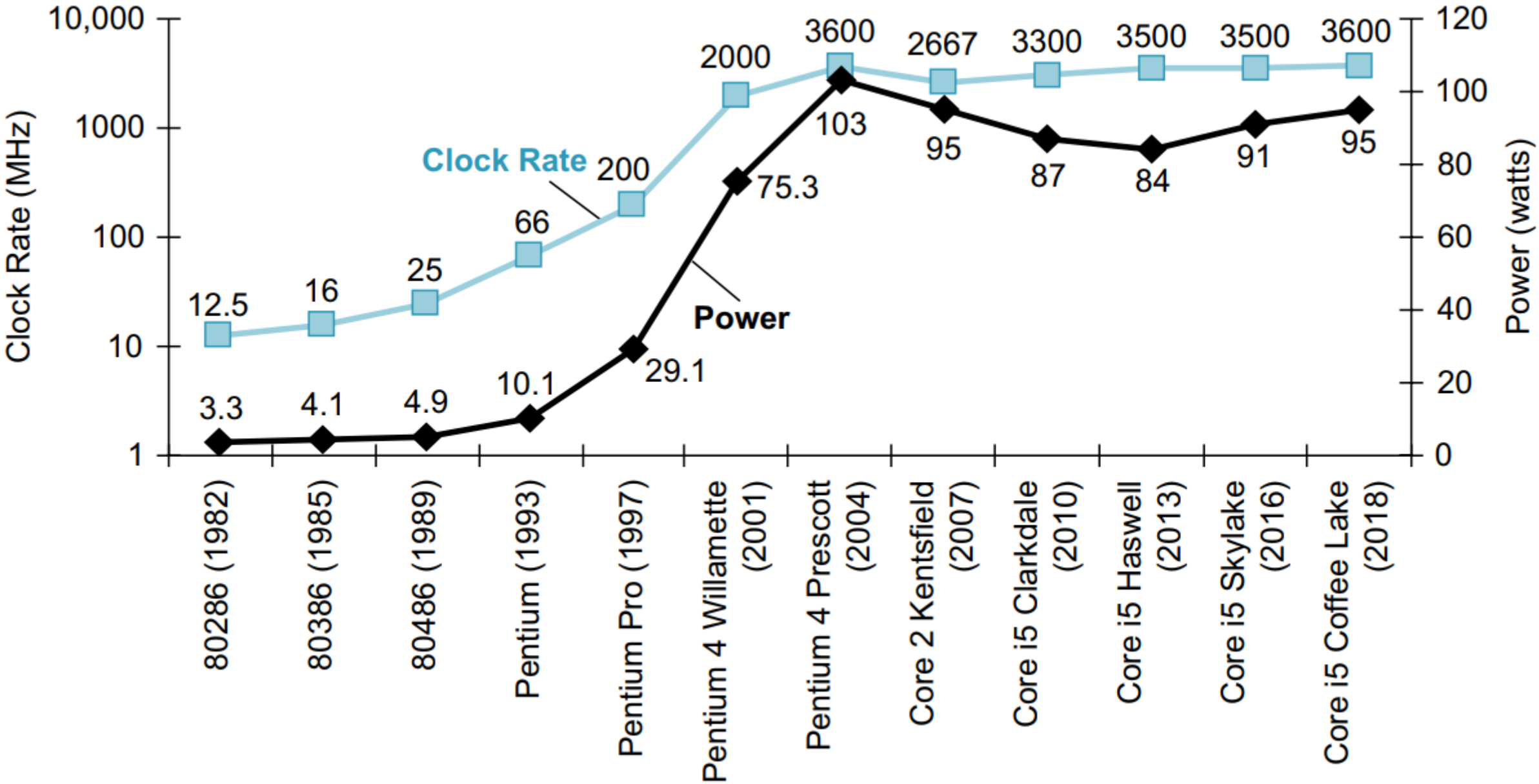
- 1965 Moore Law - he found transistor count doubling every 18 months



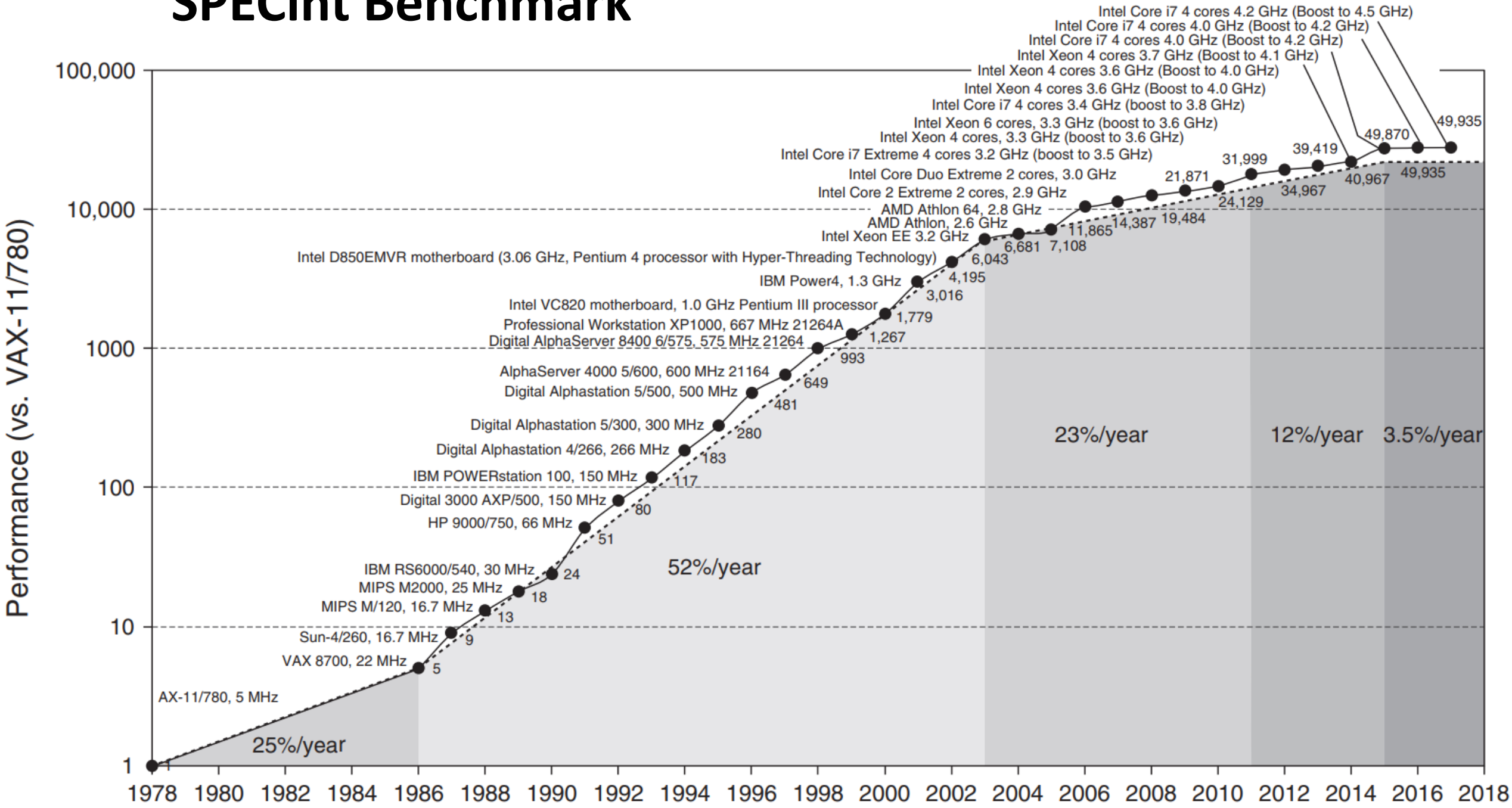
MOORE'S LAW


10,000
1,000
100
10
1
0
Clock Speed (MHz)
1970
1975
1980
1985
1990
1995
2000
2005
2010
Year
4004
8008
8080
8086
80286
Intel386
Intel486
Pentium
Pentium Pro/II/III
Pentium 4
Pentium M
Core 2 Duo

POWER WALL


Clock Rate (MHz)
10,000
1000
100
10
1
Power (watts)
120
100
80
60
40
20
0
Clock Rate
Power
12.5
16
25
66
200
2000
3600
2667
3300
3500
3500
3600
3.3
4.1
4.9
10.1
29.1
75.3
103
95
87
84
91
95
80286 (1982)
80386 (1985)
80486 (1989)
Pentium (1993)
Pentium Pro (1997)
Pentium 4 Willamette (2001)
Pentium 4 Prescott (2004)
Core 2 Kentsfield (2007)
Core i5 Clarkdale (2010)
Core i5 Haswell (2013)
Core i5 Skylake (2016)
Core i5 Coffee Lake (2018)

SPECint Benchmark



BENCHMARKING

Soru : Hangi işlemci daha hızlı?
Cevap : Neye göre, kime göre?
Çözüm : Benchmarking

Dhrystone: 1984 Reinhold Weicker tarafından geliştirildi. Floating-point operasyonlar içermiyor, yalnızca integer. DEC VAX 11/780 (1977) minicomputer performansına referansla DMIPS (Dhrystone Millions Instructions per Second), MIPS veya DMIPS/MHz olarak sonuç verilir. Örnek PicoRV32 0.516 DMIPS/MHz.

CoreMark: List processing (find and sort), matrix manipulation (common matrix operations), state machine (determine if an input stream contains valid numbers), and CRC (cyclic redundancy check). It is designed to run on devices from 8-bit microcontrollers to 64-bit microprocessors. CoreMark/MHz olarak hesaplanır. Örnek: Ibex RV32IMC 2.47 CoreMark/MHz.

SPEC: 1988'de ABD'de kurulan bir organizasyon. Güncel SPEC CPU2017. Genelde akademik çalışmalarda en çok kullanılan benchmarktır.

DHRYSTONE BENCHMARK

```
  /***************/
  /* Start timer */
  /***************/

#ifdef TIMES
  times (&time_info);
  Begin_Time = (long) time_info.tms_utime;
#endif
#ifdef TIME
  Begin_Time = time ( (long *) 0);
#endif
#ifdef MSC_CLOCK
  Begin_Time = clock();
#endif

  for (Run_Index = 1; Run_Index <= Number_Of_Runs; ++Run_Index)
  {

    Proc_5();
    Proc_4();
      /* Ch_1_Glob == 'A', Ch_2_Glob == 'B', Bool_Glob == true */
    Int_1_Loc = 2;
    Int_2_Loc = 3;
    strcpy (Str_2_Loc, "DHRYSTONE PROGRAM, 2'ND STRING");
    Enum_Loc = Ident_2;
    Bool_Glob = ! Func_2 (Str_1_Loc, Str_2_Loc);
      /* Bool_Glob == 1 */
    while (Int_1_Loc < Int_2_Loc)  /* loop body executed once */
    {
      Int_3_Loc = 5 * Int_1_Loc - Int_2_Loc;
        /* Int_3_Loc == 7 */
      Proc_7 (Int_1_Loc, Int_2_Loc, &Int_3_Loc);
        /* Int_3_Loc == 7 */
      Int_1_Loc += 1;
    } /* while */
      /* Int_1_Loc == 3, Int_2_Loc == 3, Int_3_Loc == 7 */
    Proc_8 (Arr_1_Glob, Arr_2_Glob, Int_1_Loc, Int_3_Loc);
      /* Int_Glob == 5 */
```

```
Proc_2 (Int_Par_Ref)
/******************/
    /* executed once */
    /* *Int_Par_Ref == 1, becomes 4 */

One_Fifty   *Int_Par_Ref;
{
  One_Fifty  Int_Loc;
  Enumeration   Enum_Loc;

  Int_Loc = *Int_Par_Ref + 10;
  do /* executed once */
    if (Ch_1_Glob == 'A')
      /* then, executed */
    {
      Int_Loc -= 1;
      *Int_Par_Ref = Int_Loc - Int_Glob;
      Enum_Loc = Ident_1;
    } /* if */
  while (Enum_Loc != Ident_1); /* true */
} /* Proc_2 */
```

```
Proc_3 (Ptr_Ref_Par)
/******************/
    /* executed once */
    /* Ptr_Ref_Par becomes Ptr_Glob */

Rec_Pointer *Ptr_Ref_Par;

{
  if (Ptr_Glob != Null)
    /* then, executed */
    *Ptr_Ref_Par = Ptr_Glob->Ptr_Comp;
  Proc_7 (10, Int_Glob, &Ptr_Glob->variant.var_1.Int_Comp);
} /* Proc_3 */
```

```
Proc_4 () /* without parameters */
/*******/
    /* executed once */
{
  Boolean Bool_Loc;

  Bool_Loc = Ch_1_Glob == 'A';
  Bool_Glob = Bool_Loc | Bool_Glob;
  Ch_2_Glob = 'B';
} /* Proc_4 */
```

https://github.com/sifive/benchmark-dhrystone

```
    /* perform actual benchmark */
    start_time();
#if (MULTITHREAD > 1)
    if (default_num_contexts > MULTITHREAD)
    {
        default_num_contexts = MULTITHREAD;
    }
    for (i = 0; i < default_num_contexts; i++)
    {
        results[i].iterations = results[0].iterations;
        results[i].execs      = results[0].execs;
        core_start_parallel(&results[i]);
    }
    for (i = 0; i < default_num_contexts; i++)
    {
        core_stop_parallel(&results[i]);
    }
#else
    iterate(&results[0]);
#endif
    stop_time();
    total_time = get_time();
    /* get a function of the input to report */
    seedcrc = crc16(results[0].seed1, seedcrc);
    seedcrc = crc16(results[0].seed2, seedcrc);
    seedcrc = crc16(results[0].seed3, seedcrc);
    seedcrc = crc16(results[0].size, seedcrc);
```

https://github.com/eembc/coremark

```
#include "coremark.h"
/*
Topic: Description
        Matrix manipulation benchmark

        This very simple algorithm forms the basis of many more complex
algorithms.

        The tight inner loop is the focus of many optimizations (compiler as
well as hardware based) and is thus relevant for embedded processing.

        The total available data space will be divided to 3 parts:
        NxN Matrix A - initialized with small values (upper 3/4 of the bits all
zero). NxN Matrix B - initialized with medium values (upper half of the bits all
zero). NxN Matrix C - used for the result.

        The actual values for A and B must be derived based on input that is not
available at compile time.
*/
ee_s16 matrix_test(ee_u32 N, MATRES *C, MATDAT *A, MATDAT *B, MATDAT val);
ee_s16 matrix_sum(ee_u32 N, MATRES *C, MATDAT clipval);
void   matrix_mul_const(ee_u32 N, MATRES *C, MATDAT *A, MATDAT val);
void   matrix_mul_vect(ee_u32 N, MATRES *C, MATDAT *A, MATDAT *B);
void   matrix_mul_matrix(ee_u32 N, MATRES *C, MATDAT *A, MATDAT *B);
void   matrix_mul_matrix_bitextract(ee_u32 N, MATRES *C, MATDAT *A, MATDAT *B);
void   matrix_add_const(ee_u32 N, MATDAT *A, MATDAT val);

#define matrix_test_next(x)      (x + 1)
#define matrix_clip(x, y)        ((y) ? (x)&0x0ff : (x)&0x0ffff)
#define matrix_big(x)            (0xf000 | (x))
#define bit_extract(x, from, to) (((x) >> (from)) & (~(0xffffffff << (to))))
```

COREMARK BENCHMARK

SPEC BENCHMARKS

Standard Performance Evaluation Corporation

Home | Benchmarks | Tools | Results | Contact | Blog | Join Us | Search | Help

Benchmarks

- Cloud
- CPU
- Graphics/Workstations
- High Performance Computing
- Java Client/Server
- Machine Learning
- Storage
- Power
- Virtualization
- Results Search

- Submitting Results
  - Cloud/CPU/Java/Power
  - Storage/Virtualization
  - ACCEL/HPC/MPI/OMP
  - SPECapc/SPECviewperf
  - SPECworkstation

Tools

- SERT Suite

The Standard Performance Evaluation Corporation (SPEC) is a non-profit corporation formed to establish, maintain and endorse standardized benchmarks and tools to evaluate performance and energy efficiency for the newest generation of computing systems. SPEC develops benchmark suites and also reviews and publishes submitted results from our member organizations and other benchmark licensees.

What's New:

February 9, 2023: The 14th International Conference on Performance Engineering (ICPE) will be held April 15-19, 2023 in Coimbra, Portugal.

December 14, 2022: The International Standards Group (ISG) has formed the Client Efficiency Committee to oversee the establishment and development of standardized client computing benchmarks for use primarily in national and international standards organizations, governmental regulations and energy efficiency programs for client computing systems. The benchmarks will focus on application types that are often used on notebooks, desktops and workstations.

December 1, 2022: The OSG CPU Committee has released version 1.1.9 of the SPEC CPU 2017 benchmark suite. The new update includes: an initial toolset to enable limited usage on RISC-V architectures (reportable runs not supported at this time), improved automated system configuration gathering tools (sysinfo) and new Academic Pricing ($50). Please note, as of February 28, 2023, all SPEC CPU 2017 results submitted to SPEC must use version 1.1.9.

https://www.spec.org/

BİLGİSAYAR BAŞARIMI (PERFORMANCE)

Başarımın tanımı değişkenlik gösterebilir. Örnek olarak tek kullanıcıya hizmet veren bir bilgisayarın tek bir işi ne kadar zamanda bitirdiği (latency) önem arz ederken çoklu kullanıcılara hizmet veren bir sunucunun tek bir işi bitirdiği süreden ziyade birim zamanda bitirdiği iş sayısı (throughput/bandwidth) önem arz etmektedir.

Bilgisayar başarımı (performance), bir işin bitmesi için geçen toplam zamanla (execution time) ters orantılıdır.

$$\text{Performance}_X = \frac{1}{\text{Execution time}_X}$$

$$\text{Performance}_X > \text{Performance}_Y$$

$$\frac{1}{\text{Execution time}_X} > \frac{1}{\text{Execution time}_Y}$$

$$\text{Execution time}_Y > \text{Execution time}_X$$

$$\frac{\text{Performance}_X}{\text{Performance}_Y} = n \qquad \frac{\text{Performance}_X}{\text{Performance}_Y} = \frac{\text{Execution time}_Y}{\text{Execution time}_X} = n$$

X'in performansı, Y'nin "n" katıdır
X, Y'den "n" katı kadar hızlıdır

BİLGİSAYAR BAŞARIMI (PERFORMANCE)

Örnek: Eğer Bilgisayar_A bir işi 10 saniyede, Bilgisayar_B 15 saniyede bitiriyorsa, Bilgisayar_A, Bilgisayar_B'den ne kadar daha hızlıdır?

$$\frac{\text{Performance}_X}{\text{Performance}_Y} = \frac{\text{Execution time}_Y}{\text{Execution time}_X} = n$$

$$\frac{Başarım\_A}{Başarım\_B} = \frac{ÇalışmaZamanı\_B}{ÇalışmaZamanı\_A} = \frac{15}{10} = 1.5$$

Bilgisayar_A, Bilgisayar_B'nin 1.5 katı kadar hızlıdır.

BİLGİSAYAR BAŞARIMI (PERFORMANCE)

Clock Cycle (Saat Çevrimi): Sayısal sistemler (CPU da buna dahil) bir saat sinyali ile senkron olarak çalışırlar. Saat sinyalinin sıklığı, bir saniyede gerçekleştirdiği çevrim miktarı ile frekans olarak tanımlanır. Örnek olarak 1 kilohertz (kHz) frekansa sahip bir saat bir saniyede 1000 çevrim gerçekleştirebilmektedir.

Frekansın tersi (1/frekans) period (periyot) olarak tanımlanmaktadır ve bir zaman ifade etmektedir. Örnek olarak 1 kHz frekansa sahip bir saatin bir çevrimi tamamlaması 1 ms sürer.

Frekans:	**Period**:
1 Hz	1 sn
5 Hz	200 ms
1 kHz	1 ms
500 kHz	2 us
1 MHz	1 us
100 MHz	10 ns

Frekans:	**Period**:
333.3 MHz	3 ns
1 GHz	1 ns
4 GHz	250 ps
10 GHz	100 ps

BİLGİSAYAR BAŞARIMI (PERFORMANCE)

Bir programın tamamlanması için gereken CPU yürütme zamanı, bu program meydana getiren toplam saat çevrim sayısının, her bir çevrim için gereken zaman (period) ile çarpılması ile hesaplanır.

$$\text{CPU execution time for a program} = \text{CPU clock cycles for a program} \times \text{Clock cycle time}$$

$$\text{CPU execution time for a program} = \frac{\text{CPU clock cycles for a program}}{\text{Clock rate}}$$

Amaç : Program yürütme zamanını en aza indirmeye çalışmak

Çözüm : Programın yürütülmesi için gerekli CPU saat çevrim sayısını azaltmak ve saat vuruş frekansını arttırmak

Nasıl : ???

BİLGİSAYAR BAŞARIMI (PERFORMANCE)

Örnek: Bir program, 2 GHz saat frekansına sahip A bilgisayarında 10 saniyede çalışıyor. Aynı program 6 saniyede yürütecek bir B bilgisayarı tasarlamak istiyoruz. Tasarımcı, saat frekansında büyük bir artış sağlayabileceğini, fakat bunun için işlemci tasarımında da bazı değişiklikler yapılması gerektiğini söylüyor. Yapılacak bu değişiklikler, aynı programın B bilgisayarında 1.2 kat daha fazla saat çevrimi ihtiyacına neden oluyor. Bu durumda tasarımcı saat frekansı olarak ne hedeflemelidir?

$$\text{CPU execution time for a program} = \frac{\text{CPU clock cycles for a program}}{\text{Clock rate}}$$

10 saniye = saat_çevrimi(A) / 2 x 10^9 → saat_çevrimi(A) = 20 x 10^9
saat_çevrimi(B) = saat_çevrimi(A) x 1.2 = 24 x 10^9

6 saniye = saat_çevrimi(B) / freq(B) → 6 saniye = 24 x 10^9 / freq(B)
freq(B) = 24 x 10^9 / 6 = 4 GHz

BİLGİSAYAR BAŞARIMI (PERFORMANCE)

Yürütme zamanı formülünü biraz açalım:

$$\text{CPU execution time for a program} = \text{CPU clock cycles for a program} \times \text{Clock cycle time}$$

$$\text{CPU clock cycles} = \text{Instructions for a program} \times \text{Average clock cycles per instruction}$$

Derleyicinin (compiler) bir program için oluşturduğu buyruklar (instruction) önem arz etmektedir. Bir programın yürütme zamanını etkileyen önemli faktörlerden bir tanesi de derleyicinin başarımıdır.

Aşağıdaki ifade kesinlikle doğru mudur?
Hipotez: Derleyici, aynı fonksiyonu yerine getirecek en az sayıda buyruk üretirse başarımı maximize etmiş olur.

Burada şunu bilmemiz lazım: Bütün buyruklar eşit sayıda saat vuruşunda tamamlanıyorsa yukarıdaki ifade doğrudur. Çünkü en az sayıda buyruk, en az sayıda saat çevrimine delalet eder. Peki durum bu mudur? Yani bütün buyruklar eşit sayıda saat vuruşunda mı işlemini tamamlar?

Table 1 Cortex-M3 instruction set summary

Operation	Description	Assembler	Cycles
Move	Register	`MOV Rd, <op2>`	1
	16-bit immediate	`MOVW Rd, #<imm>`	1
	Immediate into top	`MOVT Rd, #<imm>`	1
	To PC	`MOV PC, Rm`	1 + P
Add	Add	`ADD Rd, Rn, <op2>`	1
	Add to PC	`ADD PC, PC, Rm`	1 + P
	Add with carry	`ADC Rd, Rn, <op2>`	1
	Form address	`ADR Rd, <label>`	1
Subtract	Subtract	`SUB Rd, Rn, <op2>`	1
	Subtract with borrow	`SBC Rd, Rn, <op2>`	1
	Reverse	`RSB Rd, Rn, <op2>`	1
Multiply	Multiply	`MUL Rd, Rn, Rm`	1
	Multiply accumulate	`MLA Rd, Rn, Rm`	2
	Multiply subtract	`MLS Rd, Rn, Rm`	2
	Long signed	`SMULL RdLo, RdHi, Rn, Rm`	3 to 5[a]
	Long unsigned	`UMULL RdLo, RdHi, Rn, Rm`	3 to 5[a]
	Long signed accumulate	`SMLAL RdLo, RdHi, Rn, Rm`	4 to 7[a]
	Long unsigned accumulate	`UMLAL RdLo, RdHi, Rn, Rm`	4 to 7[a]

x86 INSTRUCTION LATENCIES

Integer instructions

Instruction	Operands	Ops	Latency	Reciprocal throughput	Execution unit
Move instructions					
MOV	r,r	1	1	1/3	ALU
MOV	r,i	1	1	1/3	ALU
MOV	r8,m8	1	4	1/2	ALU, AGU
MOV	r16,m16	1	4	1/2	ALU, AGU
MOV	r32,m32	1	3	1/2	AGU
MOV	m8,r8H	1	8	1/2	AGU
MOV	m8,r8L	1	2	1/2	AGU
MOV	m16/32,r	1	2	1/2	AGU
MOV	m,i	1	2	1/2	AGU
MOV	r,sr	1	2	1	
MOV	sr,r/m	6	9-13	8	
MOVZX, MOVSX	r,r	1	1	1/3	ALU
MOVZX, MOVSX	r,m	1	4	1/2	ALU, AGU

Arithmetic instructions					
ADD, SUB	r,r/i	1	1	1/3	ALU
ADD, SUB	r,m	1	1	1/2	ALU, AGU
ADD, SUB	m,r	1	7	2.5	ALU, AGU
ADC, SBB	r,r/i	1	1	1/3	ALU
ADC, SBB	r,m	1	1	1/2	ALU, AGU
ADC, SBB	m,r/i	1	7	2.5	ALU, AGU
CMP	r,r/i	1	1	1/3	ALU
CMP	r,m	1		1/2	ALU, AGU
INC, DEC, NEG	r	1	1	1/3	ALU
INC, DEC, NEG	m	1	7	3	ALU, AGU
AAA, AAS		9	5	5	ALU
DAA		12	6	6	ALU
DAS		16	7	7	ALU
AAD		4	5		ALU0
AAM		31	13		ALU
MUL, IMUL	r8/m8	3	3	2	ALU0
MUL, IMUL	r16/m16	3	3	2	ALU0_1
MUL, IMUL	r32/m32	3	4	3	ALU0_1
IMUL	r16,r16/m16	2	3	2	ALU0
IMUL	r32,r32/m32	2	4	2.5	ALU0
IMUL	r16,(r16),i	2	4	1	ALU0
IMUL	r32,(r32),i	2	5	2	ALU0
IMUL	r16,m16,i	3		2	ALU0
IMUL	r32,m32,i	3		2	ALU0
DIV	r8/m8	32	24	23	ALU

Sonuç olarak şunu gördük ki bütün buyruklar eşit saat vuruşunda tamamlanmıyor. Öyleyse tek başına buyruk sayısı, programın gecikmesini hesaplamak için yeterli değil!

https://www.agner.org/optimize/instruction_tables.pdf

BİLGİSAYAR BAŞARIMI (PERFORMANCE)

Yürütme zamanı formülünü biraz açalım:

$$\text{CPU execution time for a program} = \text{CPU clock cycles for a program} \times \text{Clock cycle time}$$

$$\text{CPU clock cycles} = \text{Instructions for a program} \times \text{Average clock cycles per instruction}$$

Aşağıdaki ifade kesinlikle doğru mudur?
Hipotez: Derleyici, aynı fonksiyonu yerine getirecek en az sayıda buyruk üretirse başarımı maximize etmiş olur.

Bütün buyrukların eşit sürede bitmediğini öğrendik. O zaman, bir programın yürütme zamanını buyruk sayısı cinsinden hesaplamak için, ortalama olarak bir buyruğun kaç çevrim sürdüğünü hesaplayıp, bu sayıyla buyruk sayısını çarptığımızda bu programın yürütülmesi için gereken çevrim sayısını (CPU clock cycles for a program) hesaplamış oluruz.

BİLGİSAYAR BAŞARIMI (PERFORMANCE)

$$\text{CPU execution time for a program} = \text{CPU clock cycles for a program} \times \text{Clock cycle time}$$

$$\text{CPU clock cycles} = \text{Instructions for a program} \times \text{Average clock cycles per instruction}$$

Buyruk başına çevrim (Clock cycles per instruction - CPI): CPI kavramı, ortalama olarak bir buyruk için harcanan çevrim sayısını belirtir. Tabi ki derlenen programdan programa, aynı program ve derleyici içinse mikromimariden mikromimariye değişim gösterebilir.

Örnek olarak, farklı RISC-V işlemci tasarımlarında aynı buyruğun farklı çevrim sayılarına sahip olması muhtemeldir. Boruhattı derinliği ya da tasarımdaki farklılıklar CPI sayısını etkiler.

Soru: Aynı ISA'in farklı 2 implementasyonunda Bilgisayar_A 4 GHz, Bilgisayar_B 2 GHz saat frekansında çalışmaktadır. Bir program derlendikten sonra buyrukların A bilgisayarında CPI değeri 2 iken B bilgisayarında CPI değeri 1.2 olmaktadır. Bu program hangi bilgisayarda daha hızlı çalışır, hızlı olan bilgisayar diğerinden ne kadar daha hızlıdır?

BİLGİSAYAR BAŞARIMI (PERFORMANCE)

$$\text{CPU execution time for a program} = \text{CPU clock cycles for a program} \times \text{Clock cycle time}$$

$$\text{CPU clock cycles} = \text{Instructions for a program} \times \text{Average clock cycles per instruction}$$

Saat_Çevrim(A) = BuyrukSayısı*CPI(A) = BuyrukSayısı*2
Saat_Çevrim(B) = BuyrukSayısı*CPI(B) = BuyrukSayısı*1.2

YürütmeZamanı(A) = Saat_Çevrim(A)*Periyot(A) = BuyrukSayısı*2*250ps
YürütmeZamanı(B) = Saat_Çevrim(B)*Periyot(B) = BuyrukSayısı*1.2*500ps

$$\frac{Başarım(A)}{Başarım(B)} = \frac{YürütmeZamanı(B)}{YürütmeZamanı(A)} = \frac{BuyrukSayısı*1.2*500ps}{BuyrukSayısı*2*250ps} = \frac{600ps}{500ps} = 1.2$$

“A bilgisayarı, B bilgisayarından 1.2 kat daha hızlıdır”

BİLGİSAYAR BAŞARIMI (PERFORMANCE)

$$\text{CPU execution time for a program} = \text{CPU clock cycles for a program} \times \text{Clock cycle time}$$

$$\text{CPU clock cycles} = \text{Instructions for a program} \times \text{Average clock cycles per instruction}$$

$$\text{CPU time} = \text{Instruction count} \times \text{CPI} \times \text{Clock cycle time}$$

$$\text{CPU time} = \frac{\text{Instruction count} \times \text{CPI}}{\text{Clock rate}}$$

$$\frac{\text{Performance}_X}{\text{Performance}_Y} = \frac{\text{Execution time}_Y}{\text{Execution time}_X} = n$$