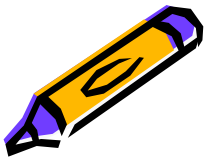
2025年度（令和7年）版

Ver. 2025-10-06a

Course number: CSC.T363

コンピュータアーキテクチャ
Computer Architecture

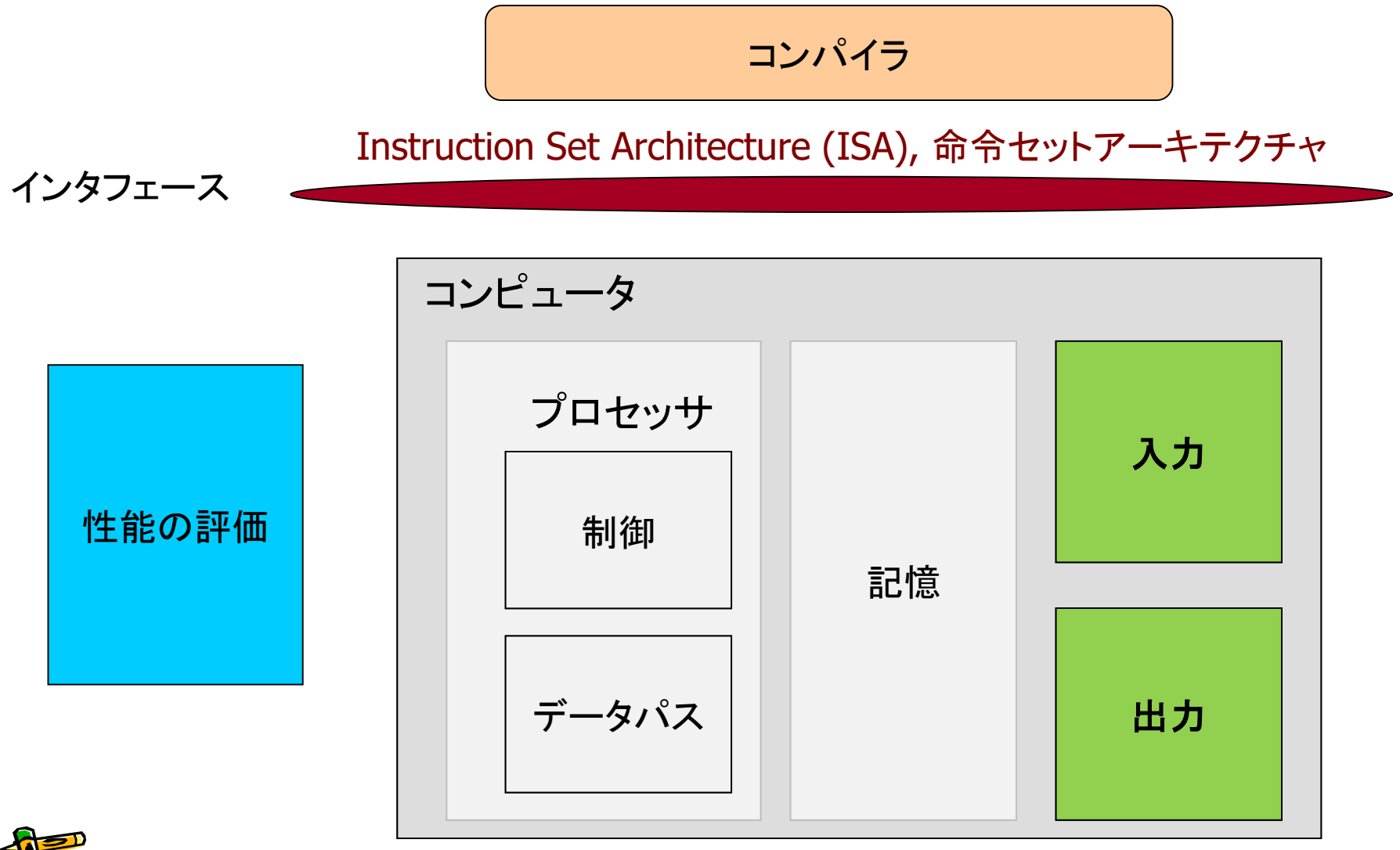
2. コンピュータの性能と消費電力の動向
Trends in Performance and Power



www.arch.cs.titech.ac.jp/lecture/CA/
Tue 13:30-15:10, 15:25-17:05
Fri 13:30-15:10

吉瀬 謙二　情報工学系
Kenji Kise, Department of Computer Science
kise _at_ c.titech.ac.jp

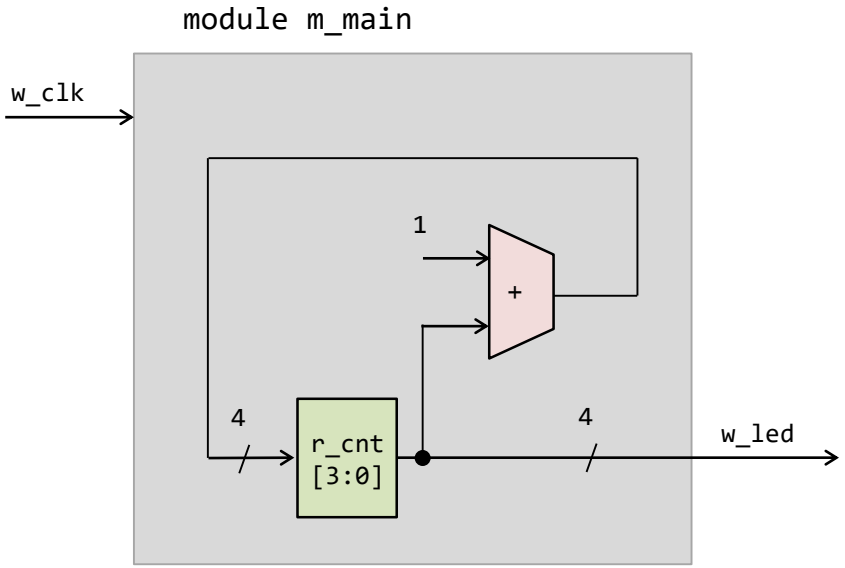
コンピュータの古典的な要素



Sequential Circuit（順序回路）とレジスタの動作

- 順序回路では、クロック信号に対するレジスタの動作を正しく理解することが重要
- レジスタは、クロック信号の立ち上がりのタイミングで入力線の値を取得(sampling)して、少し遅れて取得した値を出力線に出力する。
- レジスタの更新では、initial や always でタイミングを指定すること。
- このモジュールの波形を考える。

```
module m_main (w_clk, w_led);
  input  wire w_clk;
  output wire [3:0] w_led;

  reg [3:0] r_cnt = 0;
  always @(posedge w_clk) r_cnt <= r_cnt + 1;

  assign w_led = r_cnt;
endmodule
```



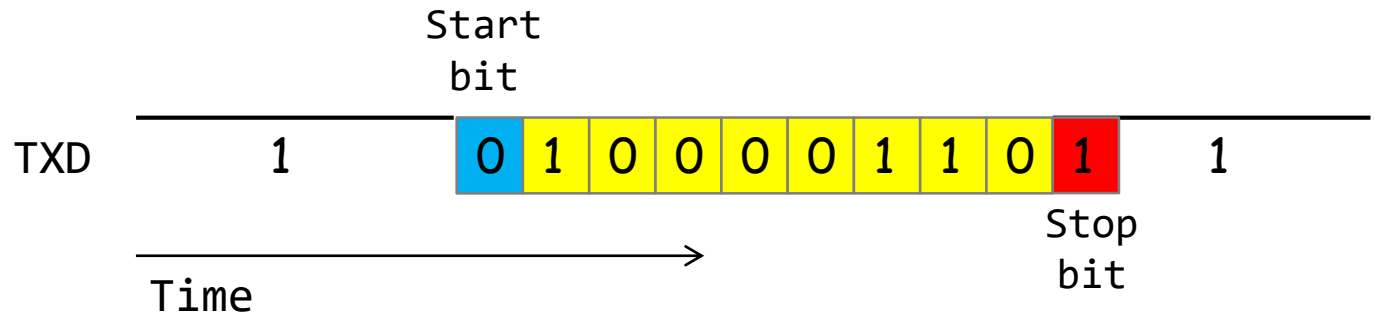
モールス符号

文字	符号	信号音	文字	符号	信号音
A	・－	Aの符号	N	－・	Nの符号
B	－・・・	Bの符号	O	－－－	Oの符号
C	－・－・	Cの符号	P	・－－・	Pの符号
D	－・・	Dの符号	Q	－－・－	Qの符号
E	・	Eの符号	R	・－・	Rの符号
F	・・－・	Fの符号	S	・・・	Sの符号
G	－－・	Gの符号	T	－	Tの符号
H	・・・・	Hの符号	U	・・－	Uの符号
I	・・	Iの符号	V	・・・－	Vの符号
J	・－－－	Jの符号	W	・－－	Wの符号
K	－・－	Kの符号	X	－・・－	Xの符号
L	・－・・	Lの符号	Y	－・－－	Yの符号
M	－－	Mの符号	Z	－－・・	Zの符号

Wikipedia

UART (Universal Asynchronous Receiver/Transmitter)

- 調歩同期方式によるシリアル信号をパラレル信号に変換したり、その逆方向の変換をおこなう集積回路をUARTと呼ぶ。8ビット(1バイト)単位でデータを送信・受信する。
- UARTを用いることで、FPGAとコンピュータの間でのお手軽なデータ通信が可能。
- 例えば、'a' という文字を送信する場合、'a' は 8'h61、8'b01100001 (次スライドのASCII Tableを参照)なので、下図のタイミングで送信線TXDを制御する.
  - データが送信されるまで送信線TXD を1とする.
  - まず、青色で示した0 (これをスタートビットと呼ぶ)を送信することで、データ送信の開始を明示.
  - 次に、黄色で示した様に送信したいデータ 8'b01100001 の最下位ビットから順番に送信する.
  - 最後に、赤色で示した1(これをストップビットと呼ぶ)を送信する.
- 1ビットを送受信するための時間間隔は送信側と受信側で同じレートを用いる. これをボー・レート (baud) と呼ぶ. 例えば、1000 baud であれば、1ビット送信の間隔は 1msec となる.



ASCII Table

Dec	Hex	Oct	Char	Dec	Hex	Oct	Char	Dec	Hex	Oct	Char	Dec	Hex	Oct	Char
0	0	0		32	20	40	[space]	64	40	100	@	96	60	140	`
1	1	1		33	21	41	!	65	41	101	A	97	61	141	a
2	2	2		34	22	42	"	66	42	102	B	98	62	142	b
3	3	3		35	23	43	#	67	43	103	C	99	63	143	c
4	4	4		36	24	44	$	68	44	104	D	100	64	144	d
5	5	5		37	25	45	%	69	45	105	E	101	65	145	e
6	6	6		38	26	46	&	70	46	106	F	102	66	146	f
7	7	7		39	27	47	'	71	47	107	G	103	67	147	g
8	8	10		40	28	50	(	72	48	110	H	104	68	150	h
9	9	11		41	29	51	)	73	49	111	I	105	69	151	i
10	A	12		42	2A	52	*	74	4A	112	J	106	6A	152	j
11	B	13		43	2B	53	+	75	4B	113	K	107	6B	153	k
12	C	14		44	2C	54	,	76	4C	114	L	108	6C	154	l
13	D	15		45	2D	55	-	77	4D	115	M	109	6D	155	m
14	E	16		46	2E	56	.	78	4E	116	N	110	6E	156	n
15	F	17		47	2F	57	/	79	4F	117	O	111	6F	157	o
16	10	20		48	30	60	0	80	50	120	P	112	70	160	p
17	11	21		49	31	61	1	81	51	121	Q	113	71	161	q
18	12	22		50	32	62	2	82	52	122	R	114	72	162	r
19	13	23		51	33	63	3	83	53	123	S	115	73	163	s
20	14	24		52	34	64	4	84	54	124	T	116	74	164	t
21	15	25		53	35	65	5	85	55	125	U	117	75	165	u
22	16	26		54	36	66	6	86	56	126	V	118	76	166	v
23	17	27		55	37	67	7	87	57	127	W	119	77	167	w
24	18	30		56	38	70	8	88	58	130	X	120	78	170	x
25	19	31		57	39	71	9	89	59	131	Y	121	79	171	y
26	1A	32		58	3A	72	:	90	5A	132	Z	122	7A	172	z
27	1B	33		59	3B	73	;	91	5B	133	[	123	7B	173	{
28	1C	34		60	3C	74	<	92	5C	134	\	124	7C	174	|
29	1D	35		61	3D	75	=	93	5D	135	]	125	7D	175	}
30	1E	36		62	3E	76	>	94	5E	136	^	126	7E	176	~
31	1F	37		63	3F	77	?	95	5F	137	_	127	7F	177	

シリアル通信による送信回路 m_uart_tx

- システムクロック 100MHz、1Mbaud を想定する送信回路

code201.v

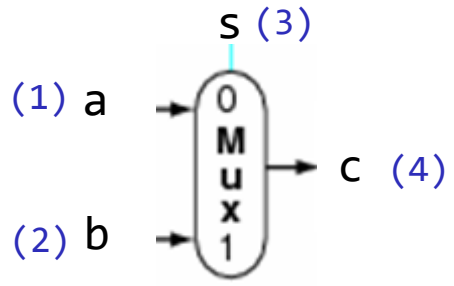
```
/* code201.v  for CSC.T363 Computer Architecture     Archlab, Institute of Science Tokyo */
/*************************************************************************************/
`default_nettype none

`define UART_CNT 100  // UART wait count, 100MHz / 100 = 1Mbaud
/*************************************************************************************/
module m_uart_tx (
    input  wire       w_clk,     // 100MHz clock signal
    input  wire       w_we,      // write enable
    input  wire [7:0] w_din,     // data in
    output wire       w_uart_tx  // UART tx, data line from FPGA to PC
);
    reg [8:0] r_buf = 9'b1_1111_1111;
    reg [7:0] r_cnt = 1;
    always @(posedge w_clk) begin
        r_cnt <= (w_we) ? 1 : (r_cnt==`UART_CNT) ? 1 : r_cnt + 1;
        r_buf <= (w_we) ? {w_din, 1'b0} : (r_cnt==`UART_CNT) ? {1'b1, r_buf[8:1]} : r_buf;
    end
    assign w_uart_tx = r_buf[0];
endmodule

/*************************************************************************************/
module m_main (
    input  wire  w_clk,       // 100MHz clock signal
    output wire  w_uart_tx    // UART tx, data line from FPGA to PC
);
    reg [31:0] r_cnt = 0;
    always @(posedge w_clk) r_cnt <= r_cnt + 1;

    wire w_we = (r_cnt[23:0]==0);
    m_uart_tx m2 (w_clk, w_we, 8'h61, w_uart_tx);
endmodule
```

code017.v モジュールのインスタンス化

- code017.v をシミュレーションして、その表示を確認すること.
- モジュール名とインスタンス名を記述して、入出力端子名を列挙する. 列挙した順序で配線が接続される.
- C言語の関数呼び出しに似ている. この例では m_mux というモジュール名のインスタンス m_mux0 を生成し、m_topモジュールの a, b, s, c をインスタンス m_mux0 の a, b, s, c に接続している.

code017.v

```
module m_top ();
  reg  a, b, s;
  wire c;
  initial begin
    #10 s <= 0; a <= 0; b <= 0;
    #10 s <= 0; a <= 0; b <= 1;
    #10 s <= 0; a <= 1; b <= 0;
    #10 s <= 0; a <= 1; b <= 1;
    #10 s <= 1; a <= 0; b <= 0;
    #10 s <= 1; a <= 0; b <= 1;
    #10 s <= 1; a <= 1; b <= 0;
    #10 s <= 1; a <= 1; b <= 1;
  end
  always@(*) #1 $display("%2d: %d %d %d -> %b", $time, s, a, b, c);
  m_mux m_mux0 (a, b, s, c);
endmodule
             (1) (2) (3) (4)
module m_mux (a, b, s, c);
  input  wire a, b, s;
  output wire c;
  assign c = s ? b : a;
endmodule
```



Simulation output

```
     s, a, b    c
11:  0  0  0 -> 0
21:  0  0  1 -> 0
31:  0  1  0 -> 1
41:  0  1  1 -> 1
51:  1  0  0 -> 0
61:  1  0  1 -> 1
71:  1  1  0 -> 0
81:  1  1  1 -> 1
```

Growth in clock rate of microprocessors

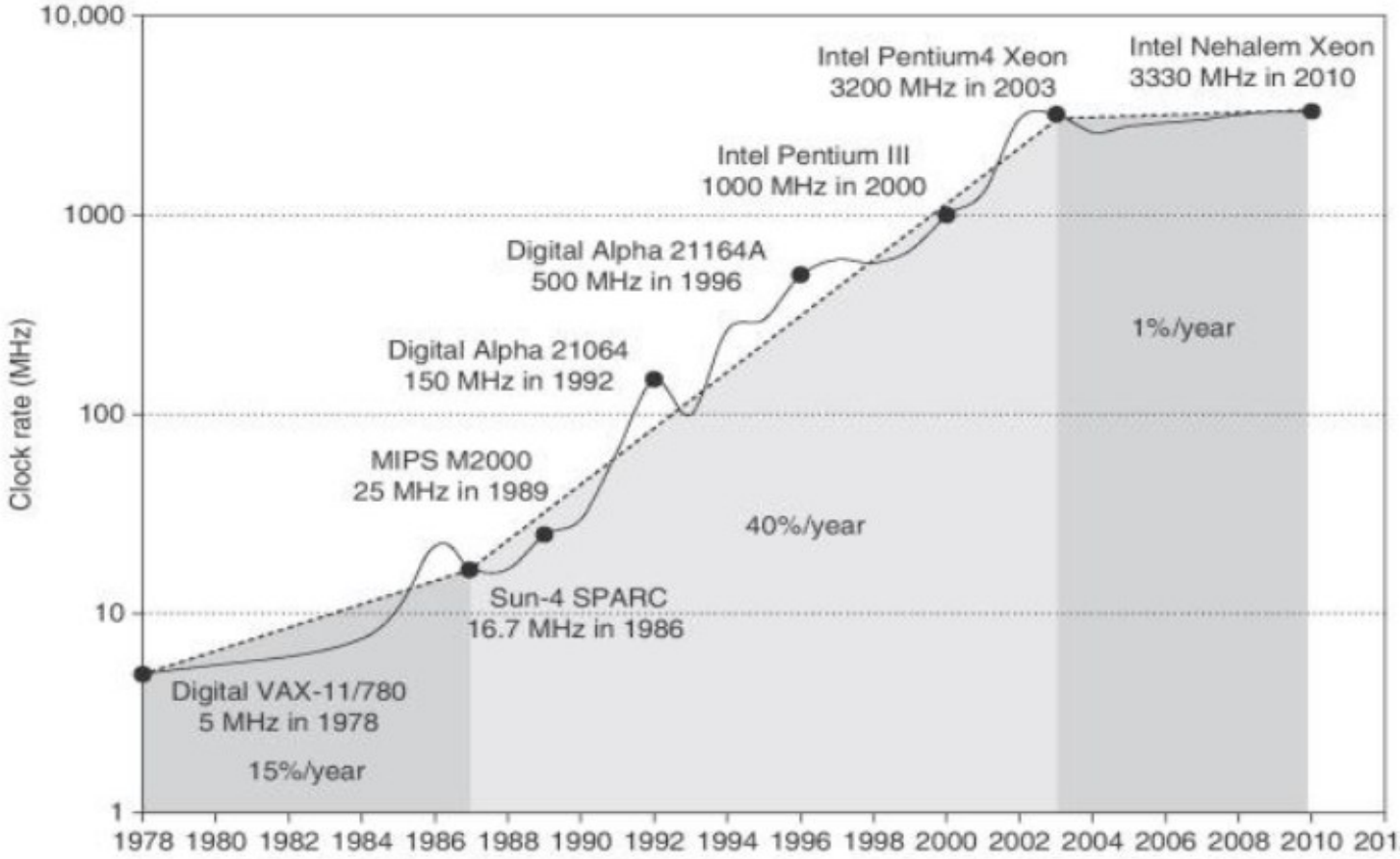
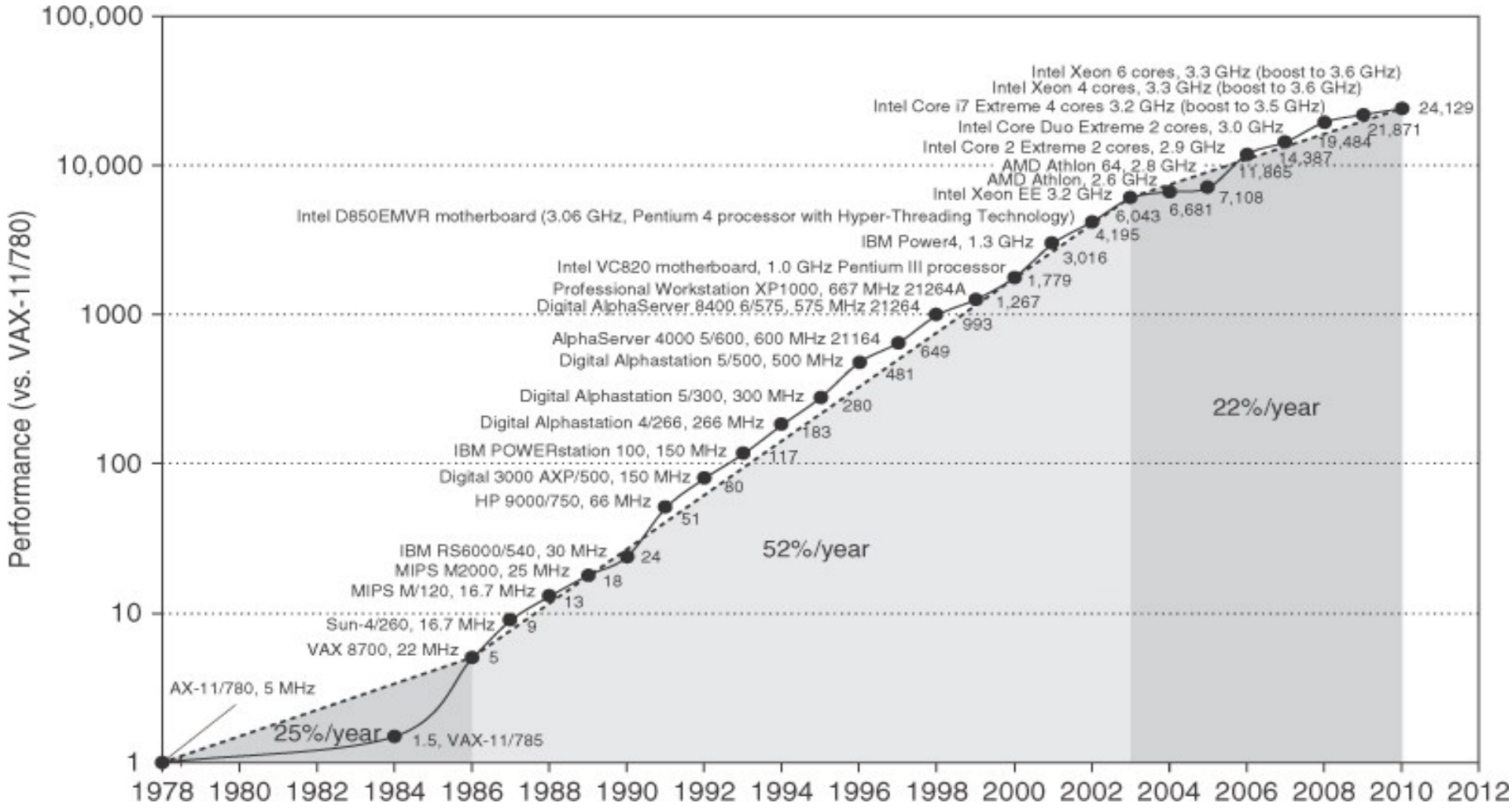


Figure 1.11 Growth in clock rate of microprocessors in Figure 1.1. Between 1978 and 1986, the clock rate improved less than 15% per year while performance improved by 25% per year. During the "renaissance period" of 52% performance improvement per year between 1986 and 2003, clock rates shot up almost 40% per year. Since then, the clock rate has been nearly flat, growing at less than 1% per year, while single processor performance improved at less than 22% per year.

From CAQA 5th edition

Growth in processor performance



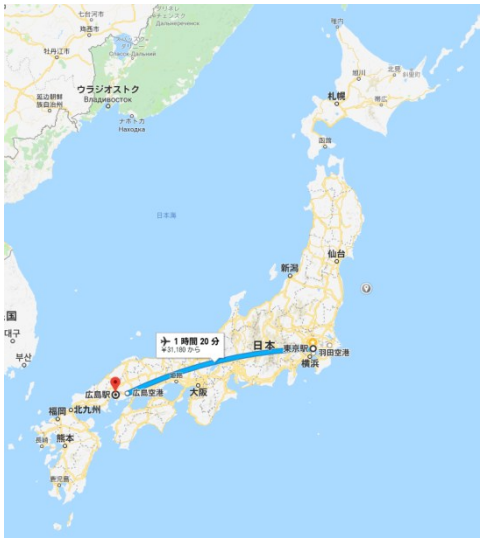
From CAQA 5th edition

Which is faster?

From Tokyo to Hiroshima

	Time Cost	Max Speed	Passengers	Throughput (P x Trips/Day)
Boeing 737	1:20 32,000yen	800km/h	170	1,530 (170 x 9)
Nozomi	4:00 18,000yen	270km/h	1,300	3,900 (1,300 x 3)

- Time to run the task (ExTime)
  - Execution time, response time, **latency**
- Tasks per day, hour, week, sec, ns … (Performance)
  - **Throughput**, bandwidth



From the lecture slide of David E Culler

Defining (Speed) Performance

Normally interested in reducing

Response time (execution time) – the time between the start and the completion of a task or a program

Important to individual users

Thus, to maximize performance, need to minimize execution time

$$\text{performance}_X = 1 / \text{execution\_time}_X$$

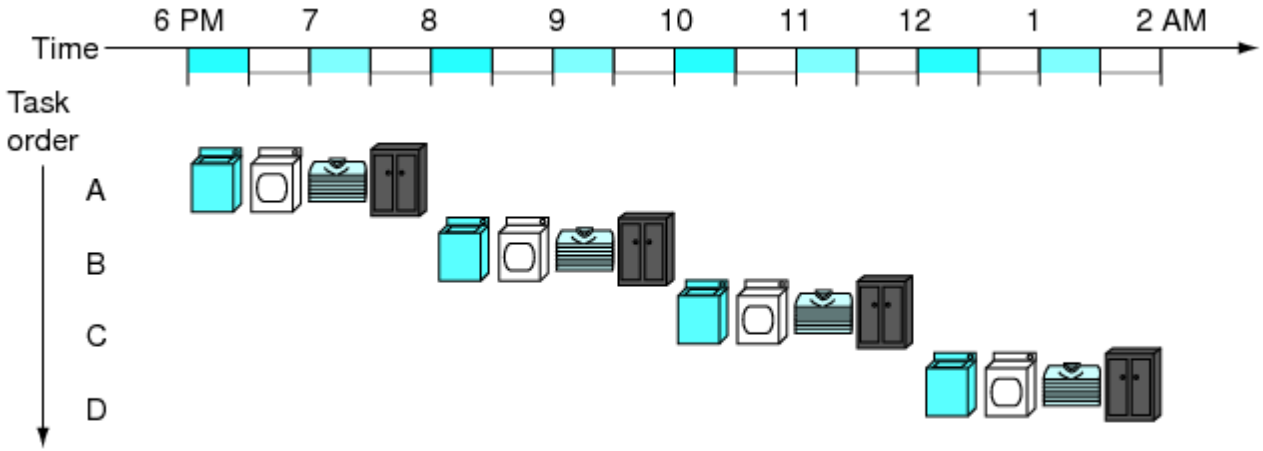
If X is n times faster than Y, then

$$\frac{\text{performance}_X}{\text{performance}_Y} = \frac{\text{execution\_time}_Y}{\text{execution\_time}_X} = n$$

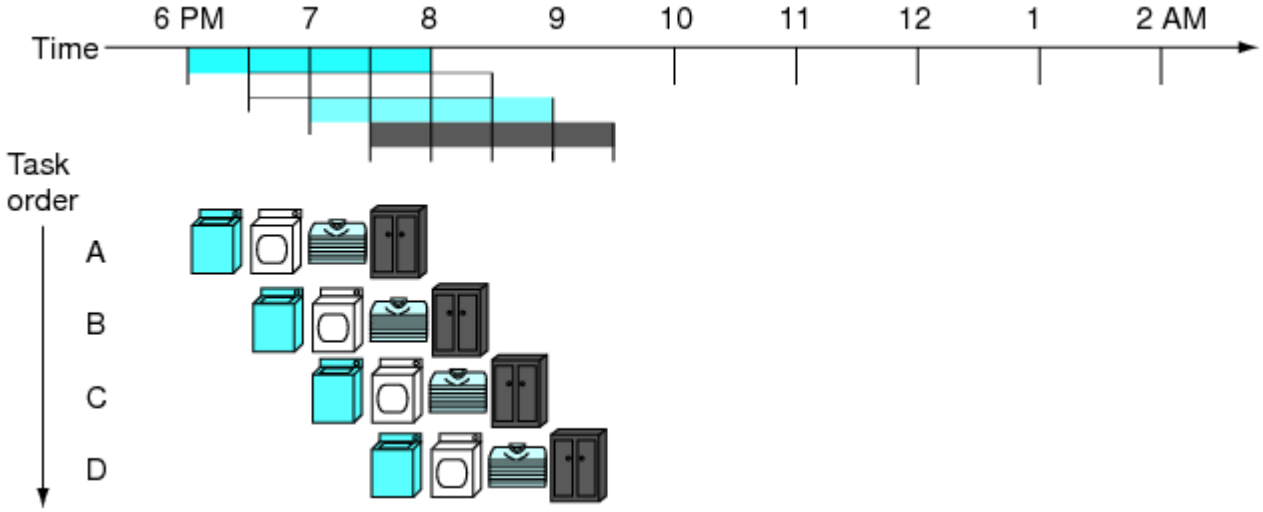
- Throughput – the total amount of work done in a given time
  - Important to data center managers
- Decreasing response time almost always improves throughput

Pipelined Processor

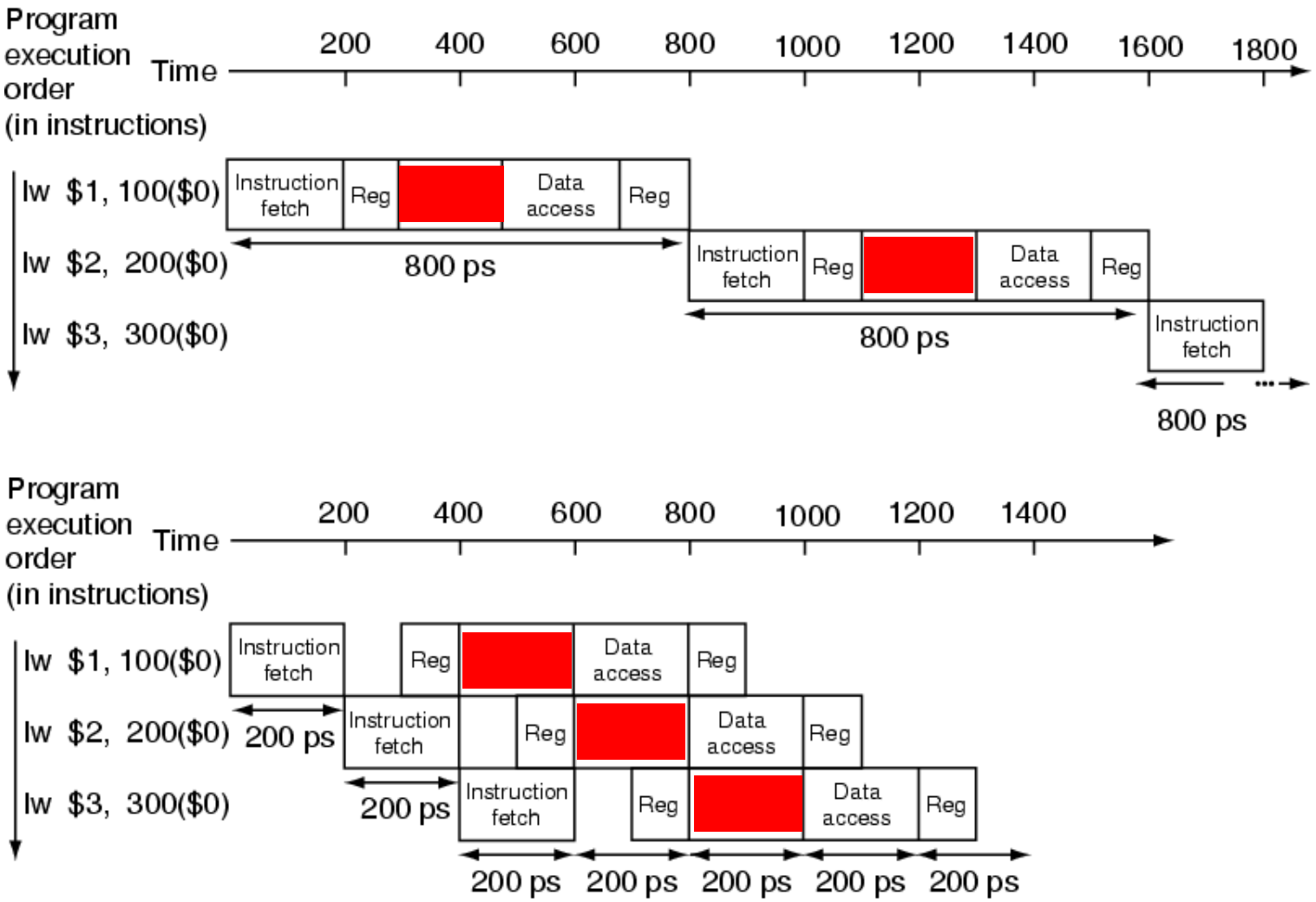
- Non pipelining (Multi-cycle)



- Pipelining



Pipelined Processor



Performance Factors

Want to distinguish elapsed time and the time spent on our task

CPU execution time (CPU time) : time the CPU spends working on a task

Does not include time waiting for I/O or running other programs

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

or

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

- Can improve performance by reducing either the length of the clock cycle or the number of clock cycles required for a program

Performance Factors

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

Performance = clock rate x 1 / # CPU clock cycles for a program

Performance = f x IPC

f: frequency, clock rate

IPC: executed (retired) instructions per cycle

```
int flag = 1;

int foo(){

    while(flag);

}
```

Pollack's Rule

- Pollack's Rule states that microprocessor "performance increase due to microarchitecture advances is roughly proportional to the square root of the increase in complexity".
  Complexity in this context means processor logic, i.e. its area.

- Superscalar, vector
  - Instruction level parallelism, data level parallelism

From multi-core era to many-core era

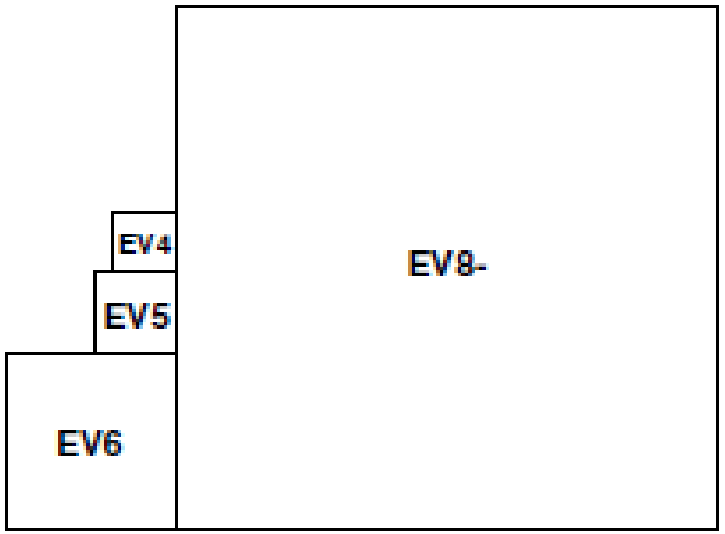


Figure 1. Relative sizes of the cores used in the study

Single-ISA Heterogeneous Multi-Core Architectures: The Potential for Processor Power Reduction, MICRO-36

Power within a microprocessor

- $Power_{dynamic}$ = 1/2 x Capacitive load x Voltage ^{2} x Frequency switched
- $P_{dynamic} = 1/2 \times C \times V^2 \times F$
  - Power required per transistor

- The first 32-bit microprocessors like Intel 80386 consumed less than two watt.
- 3.3GHz Intel Core i7 consumes 130 watts.

From multi-core era to many-core era

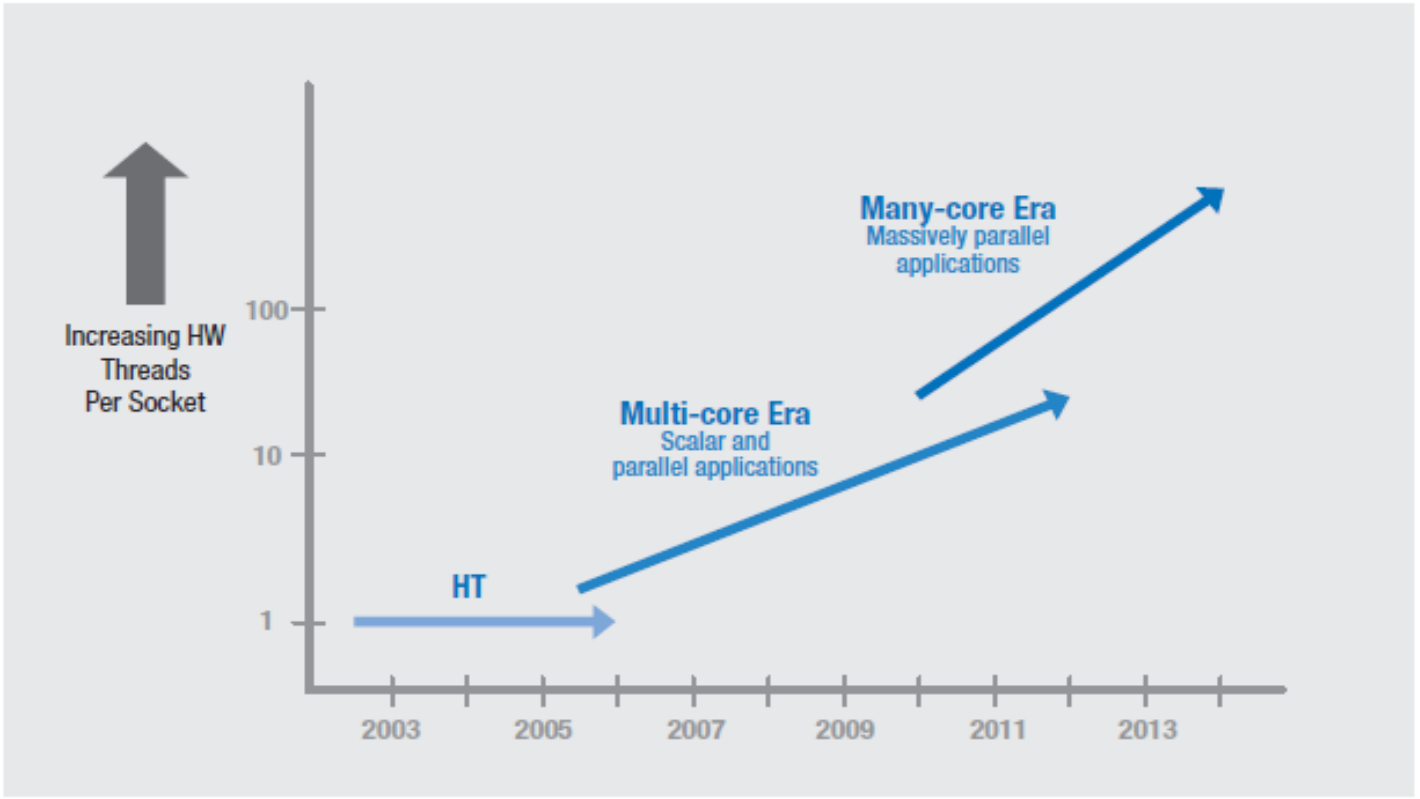
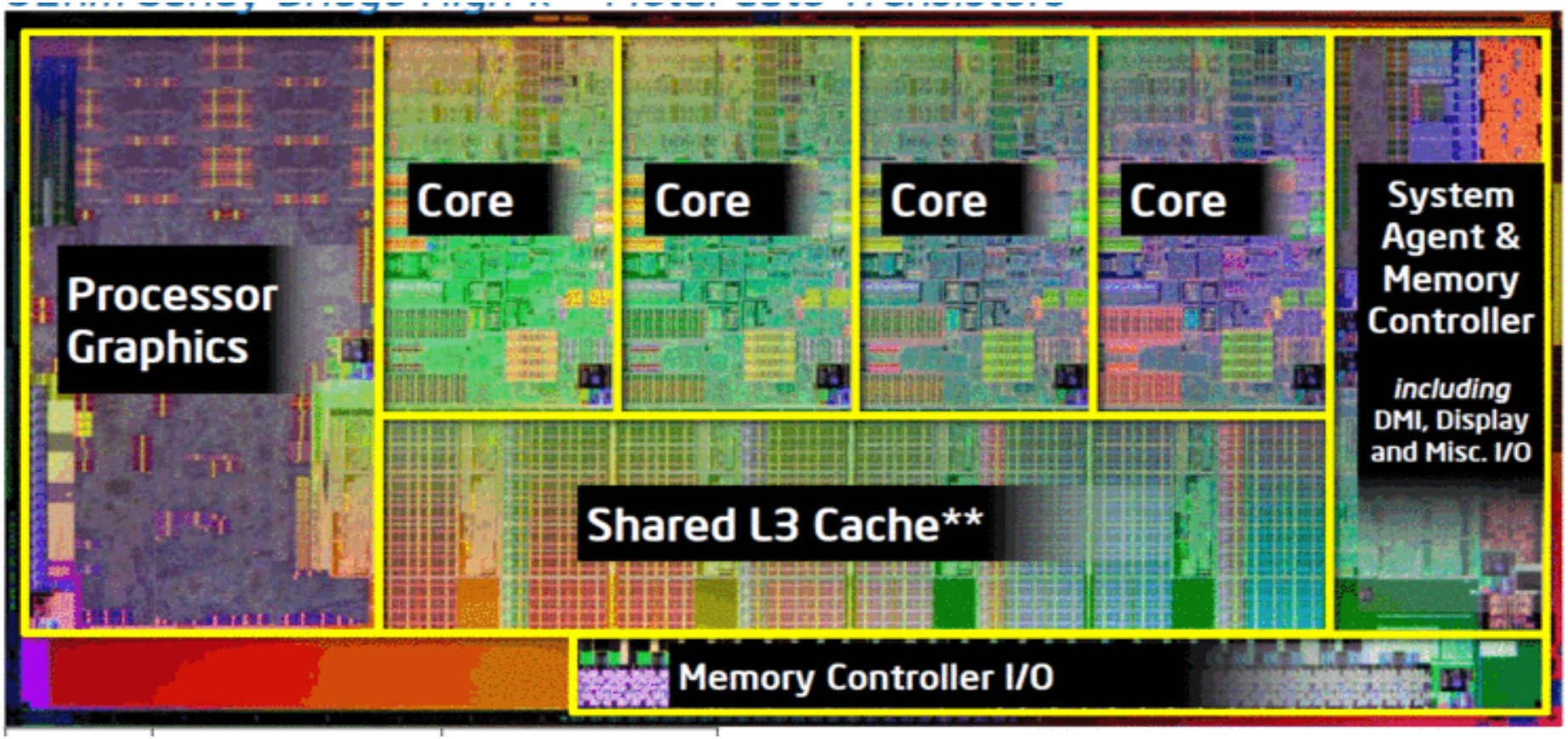


Figure 1: Current and expected eras of Intel® processor architectures

Platform 2015: Intel® Processor and Platform Evolution for the Next Decade, 2005

Intel Sandy Bridge, January 2011

4 to 8 core



アーキテクチャの異なる視点による分類

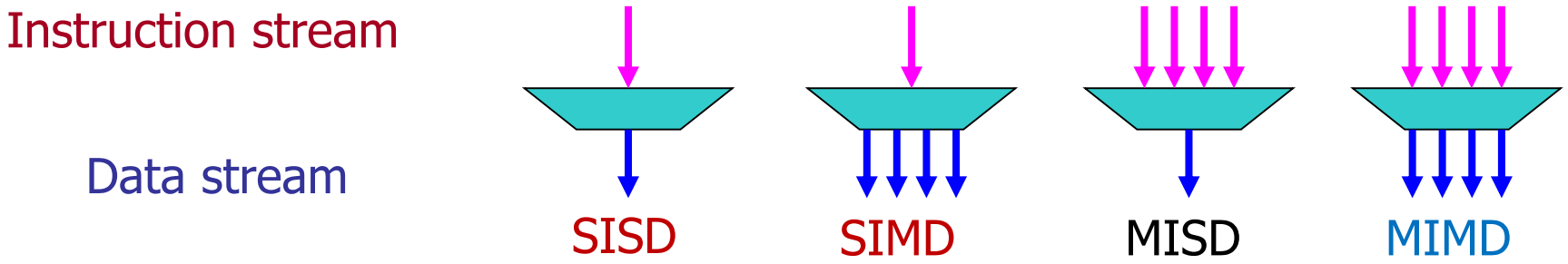
Flynnによる命令とデータの流れに注目した並列計算機の分類(1966年)

SISD (Single Instruction stream, Single Data stream)

SIMD(Single Instruction stream, Multiple Data stream)

MISD (Multiple Instruction stream, Single Data stream)

MIMD (Multiple Instruction stream, Multiple Data stream)



アーキテクチャの異なる視点による分類

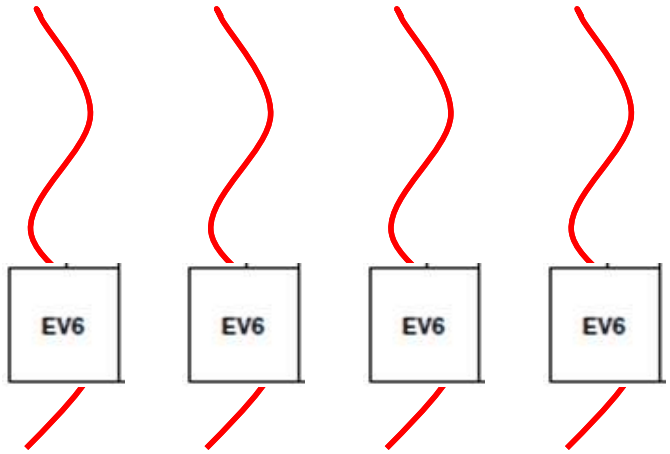
Flynnによる命令とデータの流れに注目した並列計算機の分類(1966年)

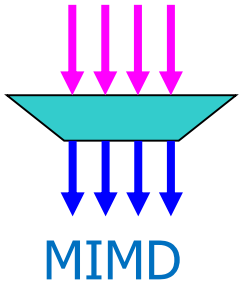
SISD (Single Instruction stream, Single Data stream)

SIMD(Single Instruction stream, Multiple Data stream)

MISD (Multiple Instruction stream, Single Data stream)

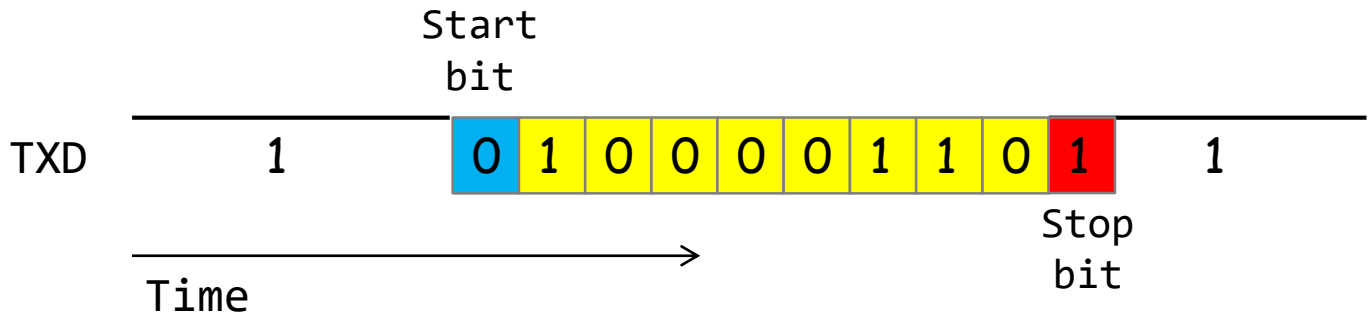
MIMD (Multiple Instruction stream, Multiple Data stream)





UART (Universal Asynchronous Receiver/Transmitter)

- 調歩同期方式によるシリアル信号をパラレル信号に変換したり、その逆方向の変換をおこなう集積回路をUARTと呼ぶ. 8ビット(1バイト)単位でデータを送信・受信する.
- UARTを用いることで、FPGAとコンピュータの間でのお手軽なデータ通信が可能.
- 例えば、'a' という文字を送信する場合、'a' は 8'h61、8'b01100001 (次スライドのASCII Tableを参照)なので、下図のタイミングで送信線TXDを制御する.
  - データが送信されるまで送信線TXD を1とする.
  - まず、青色で示した0 (これをスタートビットと呼ぶ)を送信することで、データ送信の開始を明示.
  - 次に、黄色で示した様に送信したいデータ 8'b01100001 の最下位ビットから順番に送信する.
  - 最後に、赤色で示した1(これをストップビットと呼ぶ)を送信する.
- 1ビットを送受信するための時間間隔は送信側と受信側で同じレートを用いる. これをボー・レート (baud) と呼ぶ. 例えば、1000 baud であれば、1ビット送信の間隔は 1msec となる.



シリアル通信による受信回路 m_uart_rx

- システムクロック 100MHz，1Mbaud を想定する受信回路.
- 受信した8ビットのデータを r_dout に出力し、そのことを伝えるために r_en を1にする.

code202.v

```
module m_uart_rx (
    input  wire       w_clk,   // 100MHz clock signal
    input  wire       w_rxd,   // UART rx, data line from PC to FPGA
    output wire [7:0] w_char,  // 8-bit data received
    output reg        r_en = 0 // data enable
);
    reg [2:0] r_detect_cnt = 0; /* to detect the start bit */
    always @(posedge w_clk) r_detect_cnt <= (w_rxd) ? 0 : r_detect_cnt + 1;
    wire w_detected = (r_detect_cnt>2);

    reg       r_busy = 0; // r_busy is set while receiving 9-bits data
    reg [3:0] r_bit  = 0; // the number of received bits
    reg [7:0] r_cnt  = 0; // wait count for 1Mbaud
    always@(posedge w_clk) r_cnt <= (r_busy==0) ? 1 : (r_cnt==`UART_CNT) ? 1 : r_cnt + 1;

    reg [8:0] r_data = 0;
    always@(posedge w_clk) begin
        if (r_busy==0) begin
            {r_data, r_bit, r_en} <= 0;
            if (w_detected) r_busy <= 1;
        end
        else if (r_cnt>= `UART_CNT) begin
            r_bit <= r_bit + 1;
            r_data <= {w_rxd, r_data[8:1]};
            if (r_bit==8) begin r_en <= 1; r_busy <= 0; end
        end
    end
    assign w_char = r_data[7:0];
endmodule
```