Computer Performance
JT Wunderlich PhD

1. RISC (Reduced Instruction Set Computing)
   vs CISC (Complex Instruction Set Computing)

2. Time (T) to execute a Machine Instruction

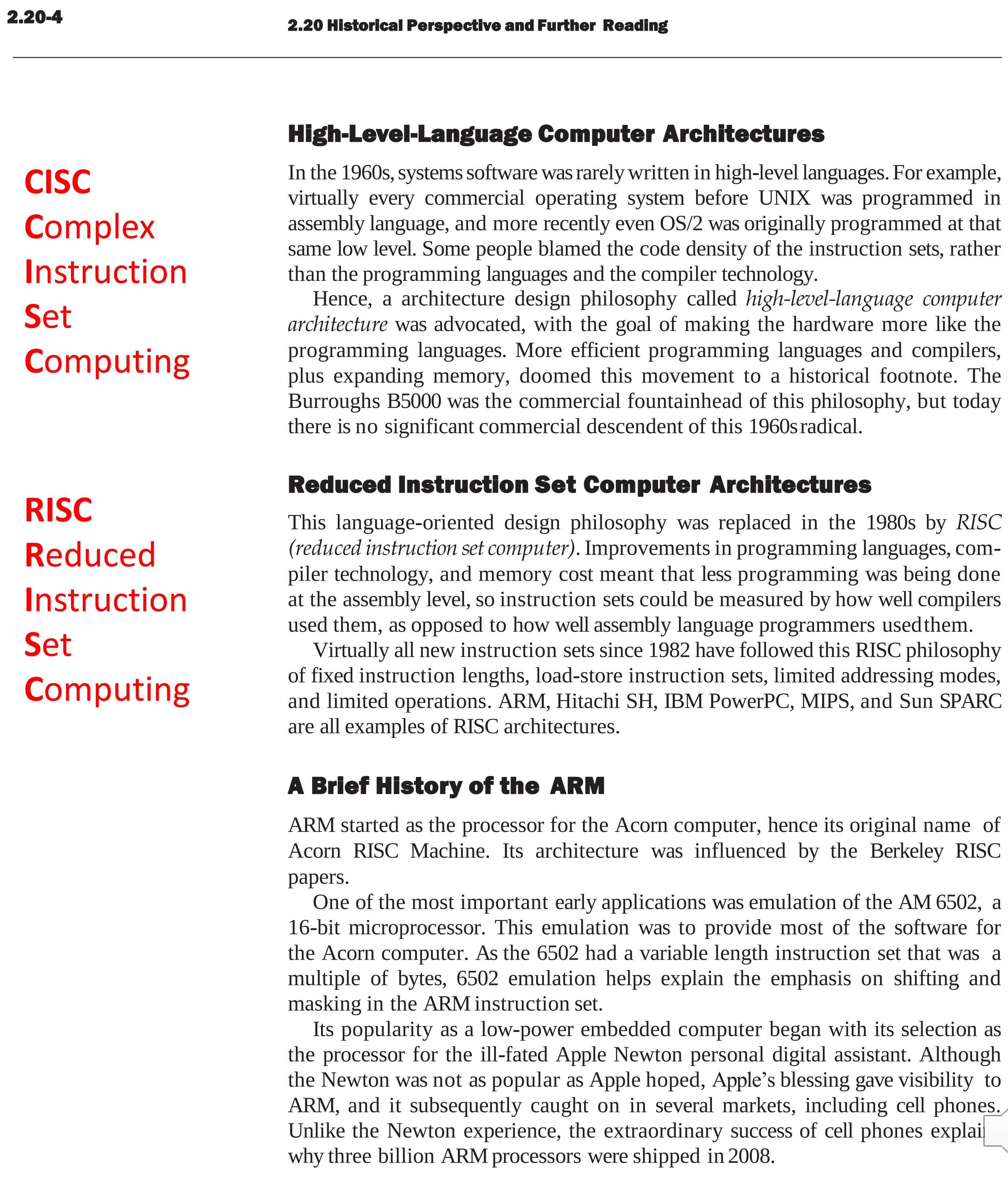
3. Ahmdahl’s Law for SPEEDUP
   - Overcoming Ahmdal’s Law

4. Mathematical Models of Speedup

5. Common Benchmarks

1. RISC (Reduced Instruction Set Computing) vs CISC (Complex Instruction Set Computing)

Review from Introductory course *(just this page from Hennesey Comp Arch text)*



2.20-4 **2.20 Historical Perspective and Further Reading**

CISC
Complex
Instruction
Set
Computing

High-Level-Language Computer Architectures

In the 1960s, systems software was rarely written in high-level languages. For example, virtually every commercial operating system before UNIX was programmed in assembly language, and more recently even OS/2 was originally programmed at that same low level. Some people blamed the code density of the instruction sets, rather than the programming languages and the compiler technology.

Hence, a architecture design philosophy called *high-level-language computer architecture* was advocated, with the goal of making the hardware more like the programming languages. More efficient programming languages and compilers, plus expanding memory, doomed this movement to a historical footnote. The Burroughs B5000 was the commercial fountainhead of this philosophy, but today there is no significant commercial descendent of this 1960s radical.

RISC
Reduced
Instruction
Set
Computing

Reduced Instruction Set Computer Architectures

This language-oriented design philosophy was replaced in the 1980s by *RISC (reduced instruction set computer)*. Improvements in programming languages, compiler technology, and memory cost meant that less programming was being done at the assembly level, so instruction sets could be measured by how well compilers used them, as opposed to how well assembly language programmers used them.

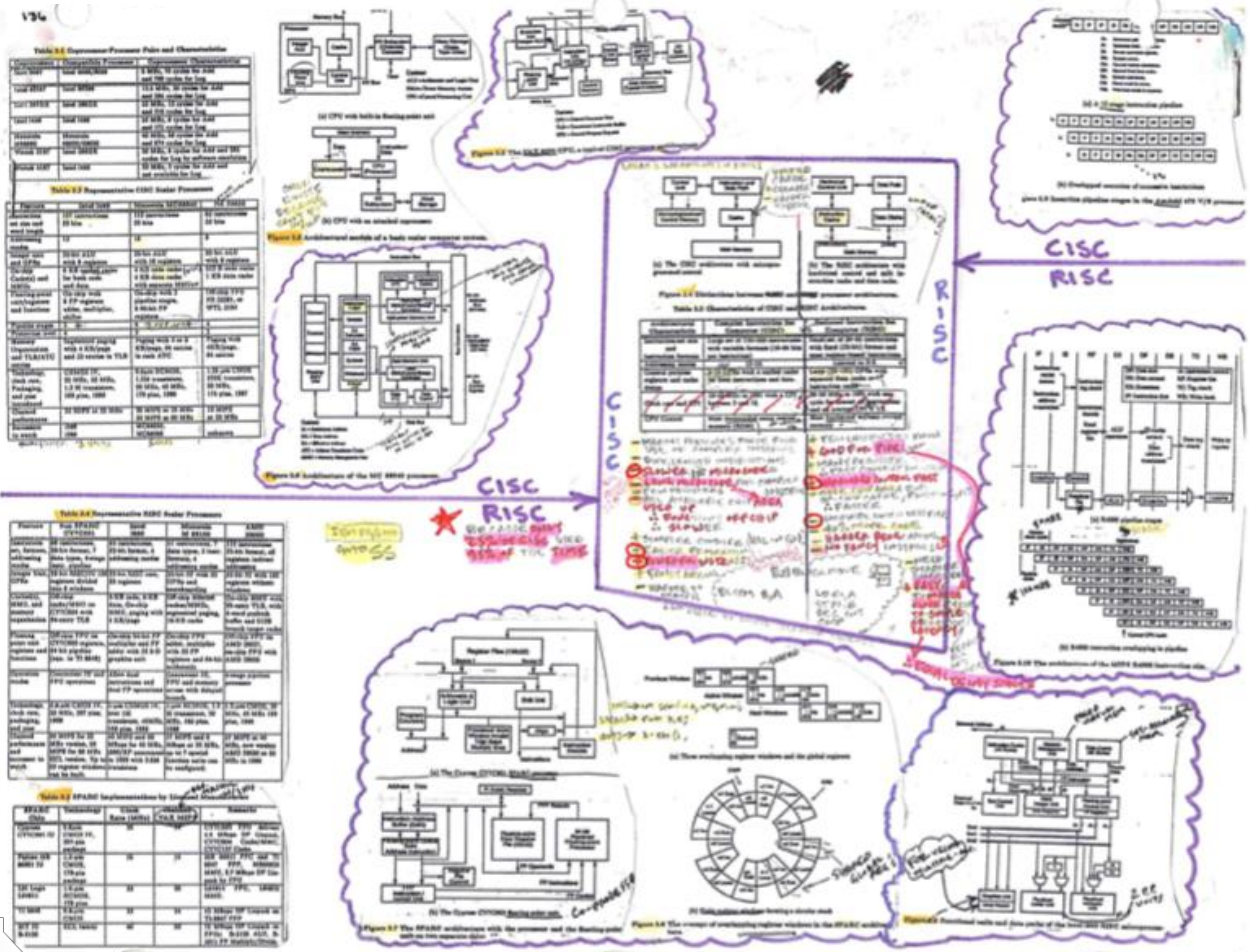
Virtually all new instruction sets since 1982 have followed this RISC philosophy of fixed instruction lengths, load-store instruction sets, limited addressing modes, and limited operations. ARM, Hitachi SH, IBM PowerPC, MIPS, and Sun SPARC are all examples of RISC architectures.

A Brief History of the ARM

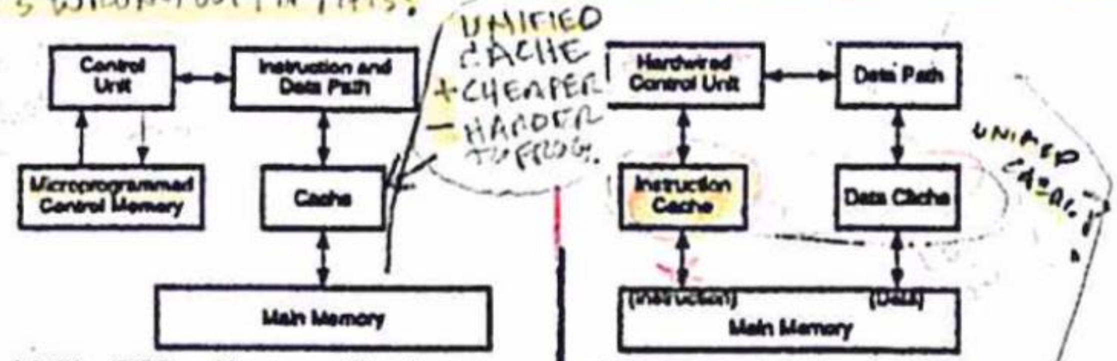
ARM started as the processor for the Acorn computer, hence its original name of Acorn RISC Machine. Its architecture was influenced by the Berkeley RISC papers.

One of the most important early applications was emulation of the AM 6502, a 16-bit microprocessor. This emulation was to provide most of the software for the Acorn computer. As the 6502 had a variable length instruction set that was a multiple of bytes, 6502 emulation helps explain the emphasis on shifting and masking in the ARM instruction set.

Its popularity as a low-power embedded computer began with its selection as the processor for the ill-fated Apple Newton personal digital assistant. Although the Newton was not as popular as Apple hoped, Apple's blessing gave visibility to ARM, and it subsequently caught on in several markets, including cell phones. Unlike the Newton experience, the extraordinary success of cell phones explai why three billion ARM processors were shipped in 2008.


CISC
RISC
CISC
RISC
C I S C
R I S C

WHAT'S WRONG WITH THIS?



(a) The CISC architecture with microprogrammed control

(b) The RISC architecture with hardwired control and split instruction cache and data cache.

Figure 3.4 Distinctions between RISC and CISC processor architectures.

RISC

Table 3.2 Characteristics of CISC and RISC Architectures

Architectural Characteristic	Complex Instruction Set Computer (CISC)	VS Reduced Instruction Set Computer (RISC)
Instruction-set size and instruction formats	Large set of 120–350 instructions with variable formats (16–64 bits per instruction)	Small set of 30–65 instructions with fixed (32-bit) format and most register-based instructions
Addressing modes	12–24	Limited to 3–5
General purpose registers and cache design	8–24 GPRs with a unified cache for both instructions and data.	Large (32–192) GPRs with separate data cache and instruction cache
Clock rate and CPI	10–33 MHz in 1991 with a CPI between 2 and 15	20–60 MHz in 1991 with one cycle for almost all instructions and an average CPI < 1.5
CPU Control	Most microcoded using control memory (ROM)	Most hardwired without control memory

CISC

− MARKET PRESSURES FORCE POOR USE OF COMPLEX INSTRUCS
− DIFF. LENGTH INSTRUCTIONS
− SLOWER IF MICROCODED
− LONG MICROCODE FOR COMPLEX INSTRUC
− FEW REGISTERS
− ALL AVAILABLE CHIP AREA USED UP
∴ FUNCTIONS OFF CHIP
∴ SLOWER
+ SIMPLER COMPILER (HLL IN SW!)
+ EASIER PROGRAMMING
+ POWERFUL INSTRUCS
+ FANCY ADDRING
− HARDER TO MATCH CLK CYCLE TO SIMPLE OPERATION LATENCY

+ FEW INSTRUC'S TO KNOW
+ GOOD FOR PIPELINE
+ MANY REGISTERS
∴ FAST CONTEXT SWITCHING
+ HARDWIRED CONTROL FAST
+ MORE CHIP AREA FOR ON-CHIP CACHE, FUNCTIONS, ETC
∴ FASTER
− HARDWIRE CONTROL LESS FLEX
− 40% MORE CODE
− HARDER PROGRAMMING
− NO FANCY INSTRUCS
− NEED SMARTER COMPILER
+ EASY TO MATCH CLOCK CYCLE TO SIMPLE OPERATION LATENCY
∴ EQUAL DELAY STAGES
BECAUSE ALL OP'S SIMPLE

EX) BLOCK MOVE

```
BLCKM B,A

LD R1,A
STR R1,B
DEC CNT
CMP
```

SHARED

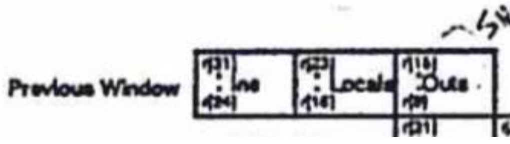
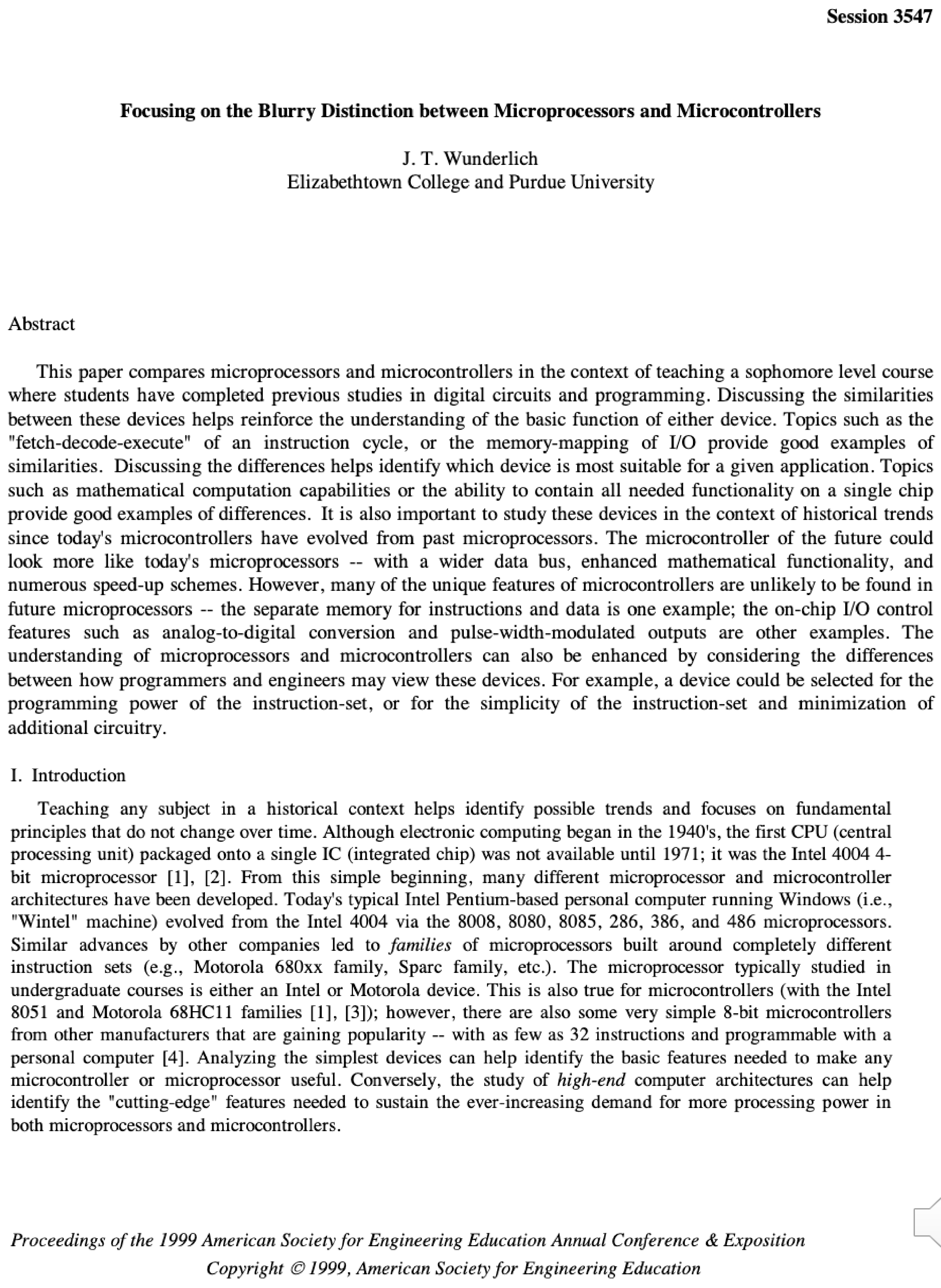


Figure 3.10

2. Time (T) to execute a Machine Instruction

Wunderlich, J.T. (1999). *Focusing on the blurry distinction between microprocessors and microcontrollers*. In Proceedings of 1999 ASEE AnnualConference & Exposition, Charlotte, NC: (session 3547), [CD-ROM]. ASEE Publications.



Session 3547

Focusing on the Blurry Distinction between Microprocessors and Microcontrollers

J. T. Wunderlich
Elizabethtown College and Purdue University

Abstract

This paper compares microprocessors and microcontrollers in the context of teaching a sophomore level course where students have completed previous studies in digital circuits and programming. Discussing the similarities between these devices helps reinforce the understanding of the basic function of either device. Topics such as the "fetch-decode-execute" of an instruction cycle, or the memory-mapping of I/O provide good examples of similarities. Discussing the differences helps identify which device is most suitable for a given application. Topics such as mathematical computation capabilities or the ability to contain all needed functionality on a single chip provide good examples of differences. It is also important to study these devices in the context of historical trends since today's microcontrollers have evolved from past microprocessors. The microcontroller of the future could look more like today's microprocessors -- with a wider data bus, enhanced mathematical functionality, and numerous speed-up schemes. However, many of the unique features of microcontrollers are unlikely to be found in future microprocessors -- the separate memory for instructions and data is one example; the on-chip I/O control features such as analog-to-digital conversion and pulse-width-modulated outputs are other examples. The understanding of microprocessors and microcontrollers can also be enhanced by considering the differences between how programmers and engineers may view these devices. For example, a device could be selected for the programming power of the instruction-set, or for the simplicity of the instruction-set and minimization of additional circuitry.

I. Introduction

Teaching any subject in a historical context helps identify possible trends and focuses on fundamental principles that do not change over time. Although electronic computing began in the 1940's, the first CPU (central processing unit) packaged onto a single IC (integrated chip) was not available until 1971; it was the Intel 4004 4-bit microprocessor [1], [2]. From this simple beginning, many different microprocessor and microcontroller architectures have been developed. Today's typical Intel Pentium-based personal computer running Windows (i.e., "Wintel" machine) evolved from the Intel 4004 via the 8008, 8080, 8085, 286, 386, and 486 microprocessors. Similar advances by other companies led to *families* of microprocessors built around completely different instruction sets (e.g., Motorola 680xx family, Sparc family, etc.). The microprocessor typically studied in undergraduate courses is either an Intel or Motorola device. This is also true for microcontrollers (with the Intel 8051 and Motorola 68HC11 families [1], [3]); however, there are also some very simple 8-bit microcontrollers from other manufacturers that are gaining popularity -- with as few as 32 instructions and programmable with a personal computer [4]. Analyzing the simplest devices can help identify the basic features needed to make any microcontroller or microprocessor useful. Conversely, the study of *high-end* computer architectures can help identify the "cutting-edge" features needed to sustain the ever-increasing demand for more processing power in both microprocessors and microcontrollers.

II. Microprocessor and Microcontroller Similarities

Discussing the similarities between microcontrollers and microprocessors helps reinforce the understanding of the basic function of either device. A good way to visualize this is by approximating a "minimal-computer-architecture" common to both devices (see Fig. 1). This figure does not make a distinction between internal (on-chip) and external (off-chip) buses or memory, code space vs. data space, etc. -- but does attempt to get across the basic idea of how these devices function.

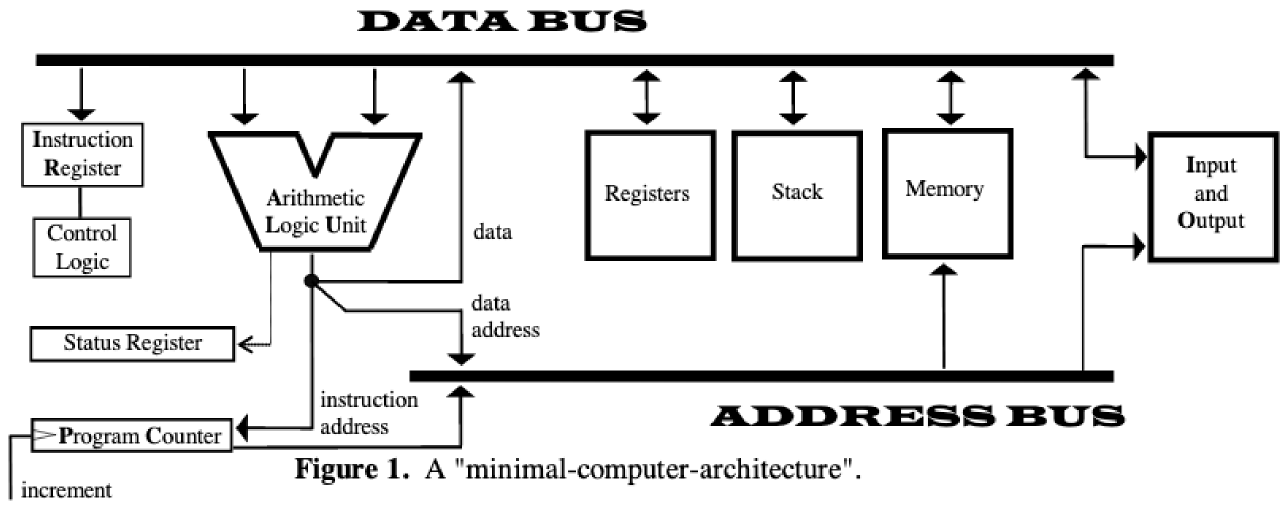


Figure 1. A "minimal-computer-architecture".

As shown in Fig. 1, each device contains:
- A program counter to address instructions to be fetched from memory.
- An instruction register to put the fetched instruction in.
- Control logic to create all routing signals after decoding the fetched instruction.
- An ALU for arithmetic and logical manipulation of data and addresses.
- Registers for storing intermediate results of calculations.
- A status register for status flags and condition codes.
- Memory for storing data and instructions.
- A stack for storing addresses (or processor status) for returning from program-calls (or interrupts).
- I/O which is addressed as memory (i.e., memory-mapped I/O).

Although the number of stages in an instruction cycle is device-dependent and can vary significantly, the fetch, decode, and execute stages are common to most microcontrollers and microprocessors. Whether instruction code is stored in internal (on-chip) or external (off-chip) ROM or RAM, it must be fetched via a data bus after being pointed to by a program counter which puts a memory address on an address bus. Both devices have instructions identified by op-codes which must be decoded during a decode phase; and each device has at least one execution phase where most control-actions are carried out. Additionally, both devices typically have a write-back phase for instructions that store ALU results into memory.

III. Microprocessor and Microcontroller Differences

Discussing the differences between microcontrollers and microprocessors helps identify which device is more suitable for a given application. A good way to examine differences is by working from the same "minimal-computer-architecture" shown above. One major difference is in how instructions and data are stored in memory, and where the memory is physically located. For typical microprocessors, all memory is located off-chip (with the exception of *some* caches) -- with a ROM used to permanently store instructions such as a bootstrap program to start-up the system, and RAM used to store data and all other instructions (i.e., "Von Neumann" architecture). For many microcontrollers, there is ROM and RAM on the chip, *address-space* for additional ROM and RAM off-chip, and registers and a stack that can be addressed as internal RAM. Also, instructions and data are stored separately (i.e., "Harvard" architecture); the ROM is used to contain all instructions upon completion of code development, and the RAM is used to store data. However, during code development the RAM can be used to store both code and data. This architecture is well suited for embedded applications where the code is fixed (i.e., burned into ROM). Having on-chip RAM also helps allow the microcontroller to be embedded as a single-chip computer.

In microprocessors, on-chip and off-chip caches (and often separate caches for data and instructions) are used to speed-up processing. These memories are made from fast static-RAM (i.e., faster, but more expensive and with less transistor-density than dynamic-RAM), and contain *most-recently-used* data and instructions (which are statistically more likely to be needed).

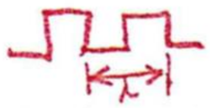
Both microcontrollers and microprocessors have a program counter to address instructions to be fetched from memory. However, the calculation of instruction addresses can be significantly more complex in microprocessors; with the target-addresses of branch instructions being prefetched using branch-prediction strategies; and with several levels of address translation required to get *real* RAM addresses from *virtual* addresses when the address space is not directly mapped to the actual available RAM.

Both devices have an instruction register to receive fetched instructions, and circuitry to decode instruction op-codes, however the control logic to create all routing signals is more complex for microprocessors which have more powerful instruction sets -- with fancier addressing modes and many instructions. The only exception to this is RISC microprocessors (i.e., Reduced Instruction-Set Computer) where intentionally simple instruction-sets and addressing modes are developed as part of the overall speed-up scheme for the processor [5]. Although microcontrollers have simple instruction-sets, they do not typically have other features needed to classify them as RISC devices (e.g., fixed instruction-length formats). They do however have on-chip RAM and several general-purpose register banks which allow faster access of data than off-chip memory-accesses -- and this has the same effect as the large register-sets found on RISC chips. Both devices have a stack for storing addresses (or processor status) for returning from program-calls (or interrupts), however microprocessors often have two stacks: a general-purpose user-stack, and a system-stack that requires *privileged* instructions to access [6].

The evolution of microprocessors (as well as more complex high-performance machines) has led to many advances in computer architecture to speed-up processing; this has included much more than increasing processor clock speed. The time to execute a program can be represented by:

$CPI = P + (m\,K)$

WHERE P = # OF CYCLES FOR INSTRUCTION DECODE + EXEC



$$T = \overline{CPI} * (I_c) * \lambda$$

where τ is the clock period in seconds per cycle (i.e., 1/*frequency*), I_c is the number of machine instructions in a given code segment, and $\overline{CPI}$ (cycles per instruction) is the average time to fetch, decode, execute, and store results for each instruction [5]. There are many strategies to decrease $\overline{CPI}$; for example, processing several instructions simultaneously (i.e., superscalar), or moving data directly between I/O and memory (i.e., Direct Memory Access). Hardware to anticipate and take "*pre*-actions" has been a design concept for many years. This not only includes prefetching data and instructions in caches, but also prefetching branch-target addresses using *Branch History Tables*, or *caching* virtual address translations using *Translation Lookaside Buffers*. Other speed-up techniques include re-ordering and optimizing instruction streams as they come into the CPU (i.e., out-of-order execution), or overlapping the individual instruction-cycle phases of many instructions (i.e., super-pipeline Many of these advances will eventually work their way into microcontroller architectures, however features designed to handle large address spaces are less likely to be needed for microcontroller applications.

m = # OF MEM ACCESSES PER INSTR.

K = RATIO

Memory Access Cycle Time

TO PROC DECODE AND EXECUTION TIME

Although both devices have an ALU for integer arithmetic and logical manipulation of data. microprocessors are much better suited for "number-crunching". and usually have a *wider* data bus and *larger* general-purpose registers to accommodate this. Microcontrollers are typically limited to 8-bit or 16-bit number representations (even though 32-bit microcontrollers are available); whereas microprocessors usually allow 32-bit representations, and contain additional floating-point hardware to allow arithmetic using much larger number ranges (and therefore much greater precision). Table 1 shows the available number range for different integer number representations.

(I.E: HOW MUCH SLOWER TO GET THINGS FROM MEMORY THAN TO [illegible] THINGS [illegible]

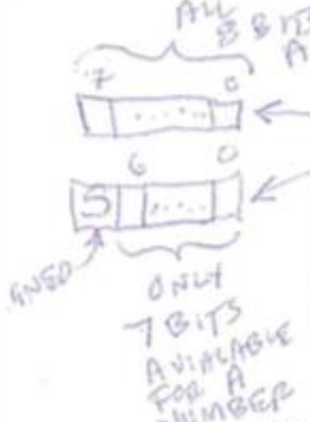

ALL 8 BITS AVAILABLE FOR A NUMBER
7
0
6
0
S
SIGNED
ONLY 7 BITS AVAILABLE FOR A NUMBER

Table 1. Number range for different integer number representations.

8-bit unsigned:	0 to (2^8)-1	=	0 to 255
8-bit signed:	-(2^8)/2 to ((2^8)/2)-1	=	-128 to 127
16-bit unsigned:	0 to (2^16)-1	=	0 to 65,535
16-bit signed:	-(2^16)/2 to ((2^16)/2)-1	=	-32,768 to 32,767
32-bit unsigned:	0 to (2^32)-1	=	0 to 4,294,967,295
32-bit signed:	-(2^32)/2 to ((2^32)/2)-1	=	-2,147,483,648 to 2,147,483,647
n-bit unsigned:	0 to (2^n)-1		
n-bit signed:	-(2^n)/2 to ((2^n)/2)-1		

One might argue that 8-bit microcontrollers are capable of manipulating large number ranges by simply parsing and manipulating large numbers using 8-bit quantities; however, as shown in Fig.'s 2 and 3, even a simple operation like decrementing a 16-bit number until it equals another 16-bit number can be much more tedious using 8-bit arithmetic. Even if the 8051 microcontroller programming effort in Fig. 3 was avoided by programming in a high-level language (e.g., C), the resulting machine code after compilation would still require many more bytes than if 16-bit registers where available (i.e., 22 vs. 8 for these examples). The limited 8051 instruction-set also contributes to the many lines of code needed in Fig. 3; the accumulator must be used to receive all results from the ALU, therefore forcing intermediate results to be moved in and out of the accumulator (i.e., at lines #8 and #11); whereas a typical microprocessor will have many register-to-register instructions, allowing intermediate results of computations to be left in several register locations (see MC68000 code in Fig. 2)[6]. Although large program size is not usually a problem for microprocessor systems, it can be a problem for microcontrollers which can have limited space for code (i.e., in ROM) -- especially if all of the code is to fit in the on-chip ROM.

← IN ORIGINAL APPLE MACHINES BEFORE IBM DEVELOPED POWERPC CHIP FOR APPLE MAC'S

Figure 2. Example MC68000 microprocessor program using 16-bit arithmetic to do a 16-bit task: Decrement the 16-bits in general-purpose data register D0 until it reaches the 16-bit number in general-purpose data register D2.

BRANCH IF EQUAL — W MEANS WORD LENGTH

```
LINE                                                                      # OF    # OF
                                                                          BYTES   CYCLES
01    check:  CMP.W D0, D2   ; compare D0 and D2, set appropriate condition flag   2       4
02            DBE D0, check  ; decrement, and jump to " check " until D0 and D2 equal   4   10 to 12
03    done:   NOP            ; program finished                                  2       4
                                                                          ---
                                                                 TOTAL =  8
```

NO OPERATION

← MAIN FOCUS OF EAR/CS 333 DIGITAL DESIGN + INTERFACING

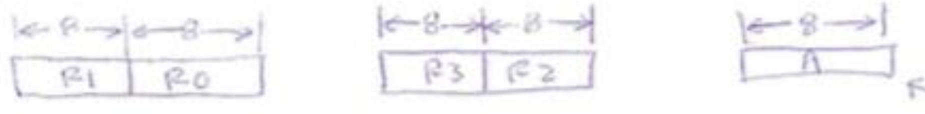
Figure 3. Example 8051 microcontroller program using 8-bit arithmetic to do a 16-bit task: Decrement the 8-bit general-purpose registers R1 and R0 as one concatenated 16-bit number until it reaches the 16-bit number made by concatenating the contents of the 8-bit general-purpose registers R3 and R2.

SAME THING IN 8051 BECAUSE REGISTERS ARE PART OF ON-CHIP RAM

MOVE — SHORT JUMP — SUBTRACT WITH BORROW — MEANS JUST # NOT HEX — IF Φ IN A

```
LINE                                                                                # OF    # OF
                                                                                    BYTES   CYCLES
00    check:   MOV A, R0           ;put low-order byte in accumulator                  1       1
01             CJNE A, 02h, dcrmnt ;conditional jump to "dcrmnt" if not equal to R2 contents   3   2
02             MOV A, R1           ;put high-order byte in accumulator                 1       1
03             CJNE A, 03h, dcrmnt ;conditional jump to " dcrmnt " if not equal to R3 contents   3   2
04             SJMP done           ;countdown finished, jump to "done"                 2       2
05    dcrmnt:  MOV A, R0           ;put low-order byte in accumulator                  1       1
06             CLR C               ;must clear carry flag since used in subtraction    1       1
07             SUBB A, #01h        ;decrement (and possibly set borrow)                2       1
08             MOV R0 ,A           ;temporarily store new high-order byte in R0        1       1
09             MOV A, R1           ;put high-order byte in accumulator                 1       1
10             SUBB A, #00h        ;subtract borrow (i.e., carry bit is set if borrow at line #07)   2   1
11             MOV R1 ,A           ;temporarily store new high-order byte in    R1     1       1
12             SJMP check          ;jump to "check "                                   2       2
13    done:    NOP                 ;program finished                                   1       1
                                                                                    ---
                                                                           TOTAL =  22
```

BORROW FLAG IS IN HERE, BUT 8051 USES C FLAG FOR B ("CY") — CARRY FLAG IS IN HERE



← ACCUMULATOR (A SPECIAL REGISTER)

STATUS OF GOOGLE IMAGE: "8051 PSW" (PROGRAM (PROCESSOR) STATUS WORD)

→ GOOGLE "8051 INSTRUCTION SET" FOR MORE INFO

One strength of microcontrollers is their on-chip RAM which allows faster memory access and therefore fewer cycles per instruction. This is illustrated in Fig.'s 4 and 5. Although the MC68000 microprocessor code of Fig. 4 requires less bytes, it must access the off-chip RAM for reading and writing the initial and final *count*; this is significantly slower than manipulating on-chip RAM. However, the overall speed of the MC68000 microprocessor code in Fig. 4 would be as fast as the 8051 microcontroller code of Fig. 5 if the difference between the initial *count* and the desired *count* was large enough (assuming equal clock speeds). The above discussion can be easily extended to a comparison of 16-bit and 32-bit devices (i.e., when doing 32-bit arithmetic).

Figure 4. Example MC68000 microprocessor program using 16-bit arithmetic to do a 16-bit task; Decrement the 16-bits at RAM location 2000h until it reaches the 16-bit number in general-purpose data register D2; then store count back into memory.

LINE				# OF BYTES	# OF CYCLES
00		MOVE.W $2000, D0	; copy original count into register D0 from RAM (off-chip)	4	12
01	check:	CMP.W D0, D2	; compare D0 and D2. set appropriate condition flag	2	4
02		DBE D0, check	; decrement, and jump to " check " <u>until</u> D0 and D2 equal	4	10 to 12
03		MOVE.W D0, $2000	; write count to RAM (off-chip) from D0	4	12
04	done:	NOP	; program finished	2	4
			TOTAL =	16	42 TO 44

Figure 5. Example 8051 microcontroller program using 8-bit arithmetic to do a 16-bit task; Decrement the 8-bit contents of internal RAM addresses 21h and 20h as one concatenated 16-bit number until it reaches the 16-bit number made by concatenating the contents of the 8-bit general-purpose registers R3 and R2 [2].

LINE				# OF BYTES	# OF CYCLES
00	check:	MOV A, 20h	;get low-order byte from on-chip RAM	2	1
01		CJNE A, 02h, dcrmnt	;conditional jump to "dcrmnt" if not equal to R2 contents	3	2
02		MOV A, 21h	;get high-order byte from on-chip RAM	2	1
03		CJNE A, 03h, dcrmnt	;conditional jump to " dcrmnt " if not equal to R3 contents	3	2
04		SJMP done	;countdown finished, jump to "done"	2	2
05	dcrmnt:	MOV A, 20h	;get low-order byte from on-chip RAM for decrementing	2	1
06		CLR C	;must clear carry flag since it is used as a borrow	1	1
07		SUBB A, #01h	;decrement (and possibly set borrow)	2	1
08		MOV 20h ,A	;store new high-order byte in on-chip RAM	2	1
09		MOV A, 21h	;get high-order byte from on-chip RAM for decrementing	2	1
10		SUBB A, #00h	;subtract borrow (i.e., carry bit is set if borrow at line #07)	2	1
11		MOV 21h ,A	;store new low-order byte in on-chip RAM	2	1
12		SJMP check	;jump to "check "	2	2
13	done:	NOP	;program finished	1	1
			TOTAL =	28	18

MUCH FASTER THAN

Probably the most defining characteristic of microcontrollers is the on-chip circuitry for interfacing with external devices. This includes watchdog timers, analog-to-digital and digital-to-analog converters (ADC's and DAC's), pulse-width-modulated (PWM) outputs for driving motors, and many counters for timing and control. These circuits are not typically found on microprocessors.

The microcontroller is best suited as an embedded single-chip computer for controlling peripheral devices, whereas the microprocessor is best suited for relatively high-speed general-purpose computing, high-precision math-intensive applications (e.g., multimedia, scientific simulations, etc.), or programming which makes use of large address spaces. However, as the number of features that can fit on a single chip increases, the microcontroller and microprocessor of the future could be packaged as a single device -- a single-chip computer with all the advantages of both, where the engineer or programmer could select which features to use. Although this might seem wasteful, mass production could make these devices inexpensive enough to justify this strategy.

COMPLEX INSTRUCTION SET COMPUTING

IV. Different Perspectives

The understanding of microprocessors and microcontrollers can also be enhanced by considering the differences between how programmers and engineers may view these devices. The complex instruction-set and addressing modes of most microprocessors might be considered a big advantage by systems-level programmers who are willing (and able) to make use of these features -- this bias may even outweigh the on-chip features of microcontrollers. For example, if a microprocessor or microcontroller needed to be chosen for an application requiring analog numbers to be read into the device, then manipulated using 16-bit arithmetic, then displayed on analog meters, the programmer might decide that the code could be most effectively written for a 16-bit microprocessor and therefore not choose a 16-bit microcontroller with built-in analog-to-digital (ADC) and digital-to-analog (DAC) converters, on-chip ROM that might fit all the code, and on-chip RAM. An engineer however might choose a 16-bit microcontroller because of the simpler instruction-set (even though more lines of program code might be required), or because the on-chip features eliminate the need to design board-level circuits to handle analog conversions and communication between the CPU and memory.

ETOWN (EGR/CS 333) AND PURDUE

V. Laboratory Experiments

Most courses in microprocessors or microcontrollers can benefit from laboratory experiments. Appendix A. lists laboratory projects that can be used for microcontroller course [8]. These labs use an 8051 microcontroller-based test computer (the PU-552), and monitor program developed at Purdue University [2], [8]. If a microprocessor-based development system is used. some of the device-control type labs can be replaced with, for example, floating-point arithmetic exercises. The programming language used for the labs in Appendix A. is mostly 8051 Assembly -- however C programming is also required for several labs. Students can also benefit from comparing code written in both C and Assembly for the same task.

GOOGLE IMAGE "FLOATING POINT" AND RECALL EGR/CS 230 NOTES

The typical undergraduate course in microprocessors or microcontrollers requires prerequisite studies in digital circuits. The lab project shown in Fig. 6 can be used in a prerequisite digital-design course to introduce the control and flow of data in a microprocessor or microcontroller.

VI. Conclusions

A good way to learn the differences between microprocessors and microcontrollers is to work from a "minimal-computer-architecture" common to both. This reinforces the understanding of basic computer architecture fundamentals. and isolates device differences so they can be easily identified. Discussing differences also helps identify which device is most suitable for a given application. Topics such as mathematical capabilities or the ability to contain all needed functionality on a single chip provide important examples of differences.

Teaching any subject in a historical context helps identify trends. and focuses on fundamental principles that do not change over time. Since the beginning of the packaged CPU in 1971, the evolution of microprocessors has led to many advances in computer architecture; many of which will eventually work their way into microcontrollers. Future microcontrollers could look like today's microprocessors -- with enhanced mathematical functionality, and numerous instruction speed-up schemes: however. some of the *peripheral-control* features of microcontrollers are unlikely to be found in future microprocessors. It may also become possible that as the number of features that can fit on a single chip increases, future microcontrollers and microprocessors could be packaged as a single device -- a single-chip computer with all the advantages of both. where the engineer or programmer could select which features to use.

The understanding of microprocessors and microcontrollers can also be enhanced by considering the differences between how programmers and engineers may view these devices. For example. a device could be selected for the programming power of the instruction-set, or for the simplicity of the instruction-set and minimization of additional circuitry.

Appendix A.

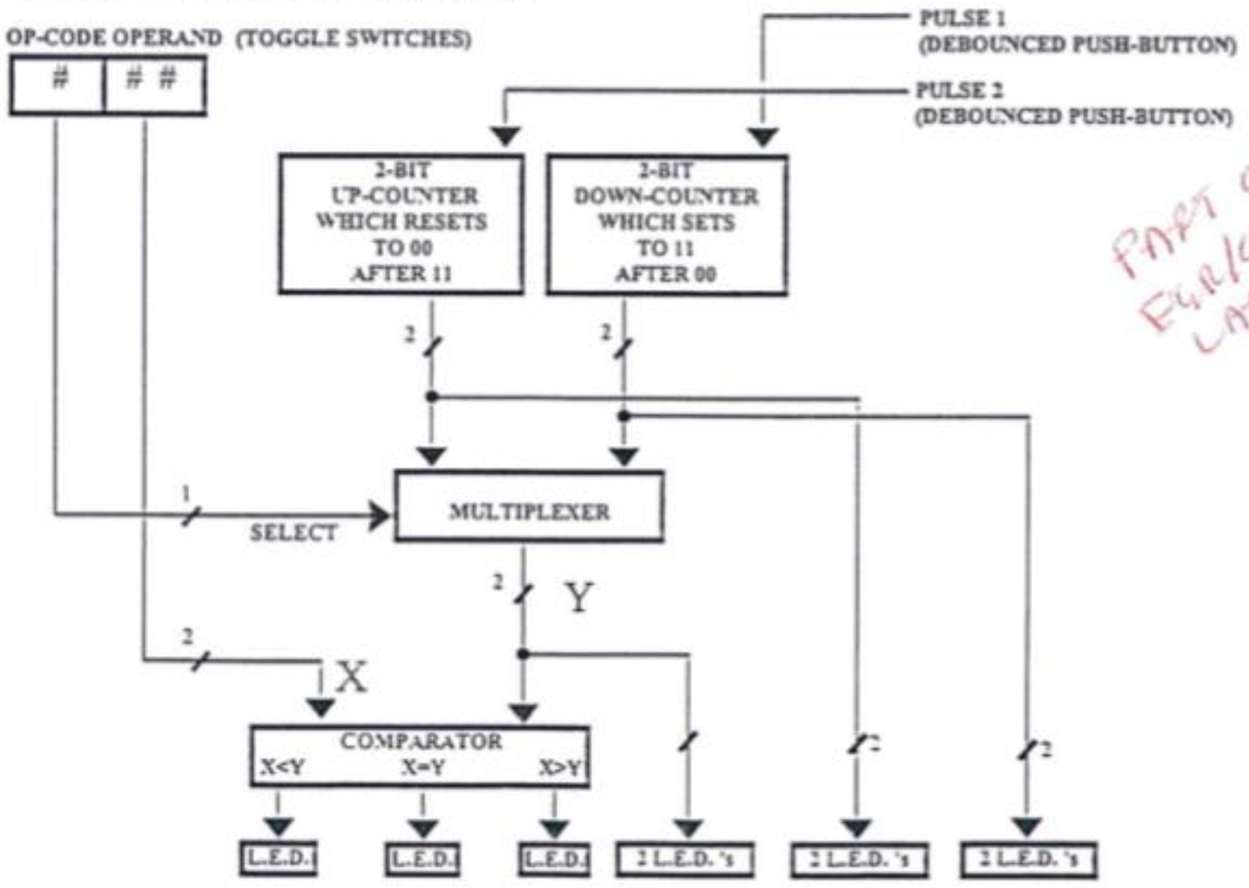
Laboratory projects for microcontroller course [8]:

1. Equipment orientation and I/O with C
2. Monitor program and program execution
3. Bus cycle timing analysis
4. Memory and I/O expansion
5. Move instructions and hand assembly
6. Branching and math instructions
7. Timing loops
8. Parallel I/O and C program development
9. Keypad and 7-segment LED displays
10. Serial I/O
11. ADC's and DAC's
12. Stepper motors and Digitalkers
13. Individual projects
14. Lab practical exam

Appendix B.

Figure 6 shows a digital-design course laboratory project to introduce the control and flow of data in a microprocessor or microcontroller. The counters are analogous to timers in a microcontroller or general-purpose registers used as counters in a microprocessor. The select line to the multiplexer can represent a *control-logic* signal generated after decoding the op-code; the comparator can represent a simplified arithmetic logic unit (ALU); and the L.E.D. circuits controlled by the comparator output can represent the contents of a status register. A variation of this lab can be made by replacing the comparator with a 2-bit parallel adder.

Instruction Set:
(OP-CODE=1): Compare operand to up-counter count
(OP-CODE=0): Compare operand to down-counter count



OP-CODE OPERAND (TOGGLE SWITCHES)
#
# #
PULSE 1 (DEBOUNCED PUSH-BUTTON)
PULSE 2 (DEBOUNCED PUSH-BUTTON)
2-BIT UP-COUNTER WHICH RESETS TO 00 AFTER 11
2-BIT DOWN-COUNTER WHICH SETS TO 11 AFTER 00
2
2
1
SELECT
MULTIPLEXER
2
Y
2
X
COMPARATOR
X<Y
X=Y
X>Y
2
2
L.E.D.
L.E.D.
L.E.D.
2 L.E.D.'s
2 L.E.D.'s
2 L.E.D.'s

PART OF ETOWN EGR/CS 433 LABS

Figure 6. A simple digital-design course laboratory project to introduce the control and flow of data in a microprocessor or microcontroller.

References

[1] K. J. Ayala, "*The 8051 Microcontroller*", 2nd ed., Minneapolis, MN: West Publishing, 1997.

[2] R. H. Barnett, "*The 8051 Family of Microcontrollers*", Englewood Cliffs, NJ: Prentice Hall, 1995.

[3] M. Kheir, "*The M68HC11 Microcontroller*", Upper Saddle River, NJ: Prentice Hall, 1997.

[4] J. B. Peatman, "*Design with PIC Microcontrollers*", Upper Saddle River, NJ: Prentice Hall, 1998.

[5] K. Hwang, "*Advanced Computer Architecture: Parallelism, Scalability, Programmability*", McGraw-Hill, 1993.

[6] T. L. Harman and B. Lawson, "*The Motorola 68000 Microprocessor Family*", Englewood Cliffs, NJ: Prentice Hall, 1985.

[7] G. G. Langdon, " *Computer Design*", San Jose, CA: Computeach Press, 1982.

[8] N. S. Widmer, "*Introduction to Microprocessors Laboratory Manual*", West Lafayette, IN: Learning Systems, 1998.

CPI IS FOUND BY LOOKING IN THE BACK OF A "DATA-BOOKS" FOR A PROCESSOR

→ CYCLE # GIVEN MAY:

1. BE EXPRESSED AS A RANGE WITH EXPLANATION OF WHAT EFFECTS RANGE:
   - A) CACHE HIT (I.E. DATA FOUND IN CACHE)
     OR
     " " (" NOT) ∴ HAD TO GET OUT OF RAM
   - B) IF BRANCHING TYPE INSTRUCTION ~~INSTR~~, WHETHER BRANCH "TAKEN OR NOT"
   - C) POSSIBLE DELAYS DUE TO DATA DEPENDENCES BETWEEN TWO SIMULTANEOUS INSTRUC.S IN A SUPERSCALAR PROC
   - D) . . .
2. BE EXPRESSED AS A FIXED NUMBER WITH CORRESPONDING DEDUCTS FOR EVENTS THAT SLOW DOWN PROC (e.g., CACHE MISS)

EMBEDDED APPLICATION

12.5 MHZ

Figure 2. Example MC68000 microprocessor program using 16-bit arithmetic to do a 16-bit task; Decrement the 16-bits in general-purpose data register D0 until it reaches the 16-bit number in general-purpose data register D2.

```
                                                                                  # OF    # OF
LINE                                                                              BYTES  CYCLES
01      check:  CMP.W D0, D2       ; compare D0 and D2, set appropriate condition flag     2      4
02              DBE D0, check      ; decrement, and jump to " check " until D0 and D2 equal  4   10 to 12
03      done:   NOP                ; program finished                                       2      4
                                                                                          ----
                                                                                 TOTAL =   8
```

$$T = \overline{CPI} * I_c * \lambda$$

$$\overline{CPI} = \frac{4 + (10 \text{ to } 12) + 4}{3} = 6 \text{ to } 6.67$$

$$I_c = 3$$

$$\lambda = 1/f = 1/12.5\,\text{MHz} = \frac{1 \text{ cycle}}{12.5 \times 10^{6}\,\text{Hz}} = 8 * 10^{-8} \text{ SECONDS/CYCLE}$$

$$\to T = \underset{\text{CYCLES}}{(6 \text{ to } 6.67)} * \underset{\text{INSTR}}{3} * 8 * 10^{-8} \tfrac{\text{SEC}}{\text{CYC}}$$

$T = 1.44 * 10^{-6}$ SECONDS to $1.6 * 10^{-6}$ [illegible]

$T = \overline{CPI} * I_c * \tau$ = 1/frequency

Table 1.2 Performance Factors Versus System Attributes SAME

System Attributes	Performance Factors				
	Instr. Count, I_c	Average Cycles per Instruction, CPI			Processor Cycle Time, τ
		Processor Cycles per Instruction, p	Memory References per Instruction, m	Memory-Access Latency, k	
Instruction-set Architecture	X	X			
Compiler Technology	X	X	X		
Processor Implementation and Control		X			X
Cache and Memory Hierarchy				X	X

USUALLY CS PROGRAMMERS

COMPUTER ENGINEERS

Based on Eq. 1.3, the CPU time in Eq. 1.2 can also be written as $T = I_c$ 10^{-6}/MIPS. Based on the system attributes identified in Table 1.2 and the above rived expressions, we conclude by indicating the fact that the MIPS rate of a giv computer is directly proportional to the clock rate and inversely proportional to t CPI. All four system attributes, instruction set, compiler, processor, and memory tec nologies, affect the MIPS rate, which varies also from program to program.

Throughput Rate Another important concept is related to how many program system can execute per unit time, called the *system throughput* W_s (in programs/secon In a multiprogrammed system, the system throughput is often lower than the *CP throughput* W_p defined by:

$$W_p = \frac{f}{I_c \times \text{CPI}} \quad (1.$$

Note that $W_p = (\text{MIPS}) \times 10^6 / I_c$ from Eq. 1.3. The unit for W_p is programs/secon The CPU throughput is a measure of how many programs can be executed per secon based on the MIPS rate and average program length (I_c). The reason why $W_s < W$ is due to the additional system overheads caused by the I/O, compiler, and OS wh multiple programs are interleaved for CPU execution by multiprogramming or tim sharing operations. If the CPU is kept busy in a perfect program-interleaving fashio then $W_s = W_p$. This will probably never happen, since the system overhead oft causes an extra delay and the CPU may be left idle for some cycles.

MICROCONTROLLER

EXCERPT

A.0 Introduction

Appendix A lists two arrangements of mnemonics for the 8051: by f
tion, and alphabetically. The mnemonic definitions here differ from tho
the original manufacturer (Intel Corporation) in the names used for addre
or data; for example, add is used to represent an address in internal R
whereas Intel uses the name direct. The author believes that the names
here are clearer than those used by Intel. This appendix also includes an al
betical listing of the mnemonics using Intel names. There is no difference
tween the mnemonics when real numbers replace the names. For exam
MOV add,#n and MOV direct,#data become MOV 10h,#40h when the number
replaces the internal RAM address (add/direct) and 40h replaces the num
(#n/# data).

REGISTER TRANSFER NOTATION

R.T.N. ☆ DO THIS IN LAB REPORTS ☆

A.1 Mnemonics, Arranged by Function

ACCUMULATOR REGISTER

BAD! NEED A FOR EVERYTHING.

Arithmetic

Mnemonic	Description	Bytes	Cycles	Flags
ADD A,Rr	$A+Rr \rightarrow A$	1	1	C OV
ADD A,add	$A+(add) \rightarrow A$	2	1	C OV
ADD A,@Rp	$A+(Rp) \rightarrow A$	1	1	C OV AC
ADD A,#n	$A+n \rightarrow A$	2	1	C OV AC
ADDC A,Rr	$A+Rr+C \rightarrow A$	1	1	C OV AC
ADDC A,add	$A+(add)+C \rightarrow A$	2	1	C OV AC
ADDC A,@Rp	$A+(Rp)+C \rightarrow A$	1	1	C OV AC
ADDC A,#n	$A+n+C \rightarrow A$	2	1	C OV AC
DA A	$Abin \rightarrow Adec$ ONLY WORKS FOR ADDITION	1	1	C
DEC A	$A-1 \rightarrow A$	1	1	
DEC Rr	$Rr-1 \rightarrow Rr$	1	1	
DEC add	$(add)-1 \rightarrow (add)$	2	1	
DEC @Rp	$(Rp)-1 \rightarrow (Rp)$	1	1	
DIV AB	$A/B \rightarrow AB$	1	4	0 OV
INC A	$A+1 \rightarrow A$	1	1	
INC Rr	$Rr+1 \rightarrow Rr$	1	1	
INC add	$(add)+1 \rightarrow (add)$	2	1	
INC @Rp	$(Rp)+1 \rightarrow (Rp)$	1	1	
INC DPTR	$DPTR+1 \rightarrow DPTR$	1	2	
MUL AB	$A \times B \rightarrow AB$	1	4	0 OV
SUBB A,Rr	$A-Rr-C \rightarrow A$	1	1	C OV AC
SUBB A,add	$A-(add)-C \rightarrow A$	2	1	C OV AC
SUBB A,@Rp	$A-(Rp)-C \rightarrow A$	1	1	C OV AC
SUBB A,#n	$A-n-C \rightarrow A$	2	1	C OV AC

ADJUST SUM OF TWO BCD NUMBERS

8051 ☆ USES CARRY FLAG AS A BORROW FLAG

Motorola 68000 Microprocessor Expert

APPENDIX IVD: INSTRUCTION EXECUTION TIMES

D.1 INTRODUCTION

This Appendix contains listings of the instruction execution times in terms of external clock (CLK) periods. In this data, it is assumed that both memory read and write cycle times are four clock periods. A longer memory cycle will cause the generation of wait states which must be added to the total instruction time.

The number of bus read and write cycles for each instruction is also included with the timing data. This data is enclosed in parenthesis following the number of clock periods and is shown as: (r/w) where r is the number of read cycles and w is the number of write cycles included in the clock period number. Recalling that either a read or write cycle requires four clock periods, a timing number given as 18(3/1) relates to 12 clock periods for the three read cycles, plus 4 clock periods for the one write cycle, plus 2 cycles required for some internal function of the processor.

NOTE

The number of periods includes instruction fetch and all applicable operand fetches and stores.

D.2 OPERAND EFFECTIVE ADDRESS CALCULATION TIMING

Table D-1 lists the number of clock periods required to compute an instruction's effective address. It includes fetching of any extension words, the address computation, and fetching of the memory operand. The number of bus read and write cycles is shown in parenthesis as (r/w). Note there are no write cycles involved in processing the effective address.

Table D-1. Effective Address Calculation Times

Addressing Mode		Byte, Word	Long
	Register		
Dn	Data Register Direct	0(0/0)	0(0/0)
An	Address Register Direct	0(0/0)	0(0/0)
	Memory		
(An)	Address Register Indirect	4(1/0)	8(2/0)
(An) +	Address Register Indirect with Postincrement	4(1/0)	8(2/0)
− (An)	Address Register Indirect with Predecrement	6(1/0)	10(2/0)
d(An)	Address Register Indirect with Displacement	8(2/0)	12(3/0)
d(An, ix)*	Address Register Indirect with Index	10(2/0)	14(3/0)
xxx.W	Absolute Short	8(2/0)	12(3/0)
xxx.L	Absolute Long	12(3/0)	16(4/0)
d(PC)	Program Counter with Displacement	8(2/0)	12(3/0)
d(PC, ix)*	Program Counter with Index	10(2/0)	14(3/0)
#xxx	Immediate	4(1/0)	8(2/0)

*The size of the index register (ix) does not affect execution time.

OLD

Table 10.1. Intel486™ SX Microprocessor/Intel487™ SX Math CoProcessor Integer Clock Count Summary

INSTRUCTION	FORMAT	Cache Hit	Penalty if Cache Miss	Notes
INTEGER OPERATIONS				
MOV = Move:				
reg1 to reg2	1000100W 11 reg1 reg2	1		
reg2 to reg1	1000101w 11 reg1 reg2	1		
memory to reg	1000101w mod reg r/m	1	2	
reg to memory	1000100w mod reg r/m	1		
Immediate to reg	1100011w 11000 reg immediate data	1		
or	1011w reg immediate data	1		
Immediate to Memory	1100011w mod 000 r/m displacement immediate	1		
Memory to Accumulator	1010000w full displacement	1	2	
Accumulator to Memory	1010001w full displacement	1		
MOVSX/MOVZX = Move with Sign/Zero Extension				
reg2 to reg1	00001111 1011z11w 11 reg1 reg2	3		
memory to reg	00001111 1011z11w mod reg r/m	3	2	
PUSH = Push				
reg	11111111 11 110 reg	4		
or	01010 reg	1		
memory	11111111 mod 110 r/m	4	1	1
immediate	011010s0 immediate data	1		
PUSHA = Push All	01100000	11		
POP = Pop				
reg	10001111 11 000 reg	4	1	
or	01011 reg	1	2	
memory	10001111 mod 000 r/m	5	2	1
POPA = Pop All	01100001	9	7/15	16/32
XCHG = Exchange				
reg1 with reg2	1000011w 11 reg1 reg2	3		2
Accumulator with reg	10010 reg	3		2
Memory with reg	1000011w mod reg r/m	5		2
NOP = No Operation	10010000	1		
LEA = Load EA to Register	10001101 mod reg r/m			
no index register		1		
with index register		2		

z	Instruction
0	MOVZX
1	MOVSX

[illegible]

PIC MICROCONTROLLER

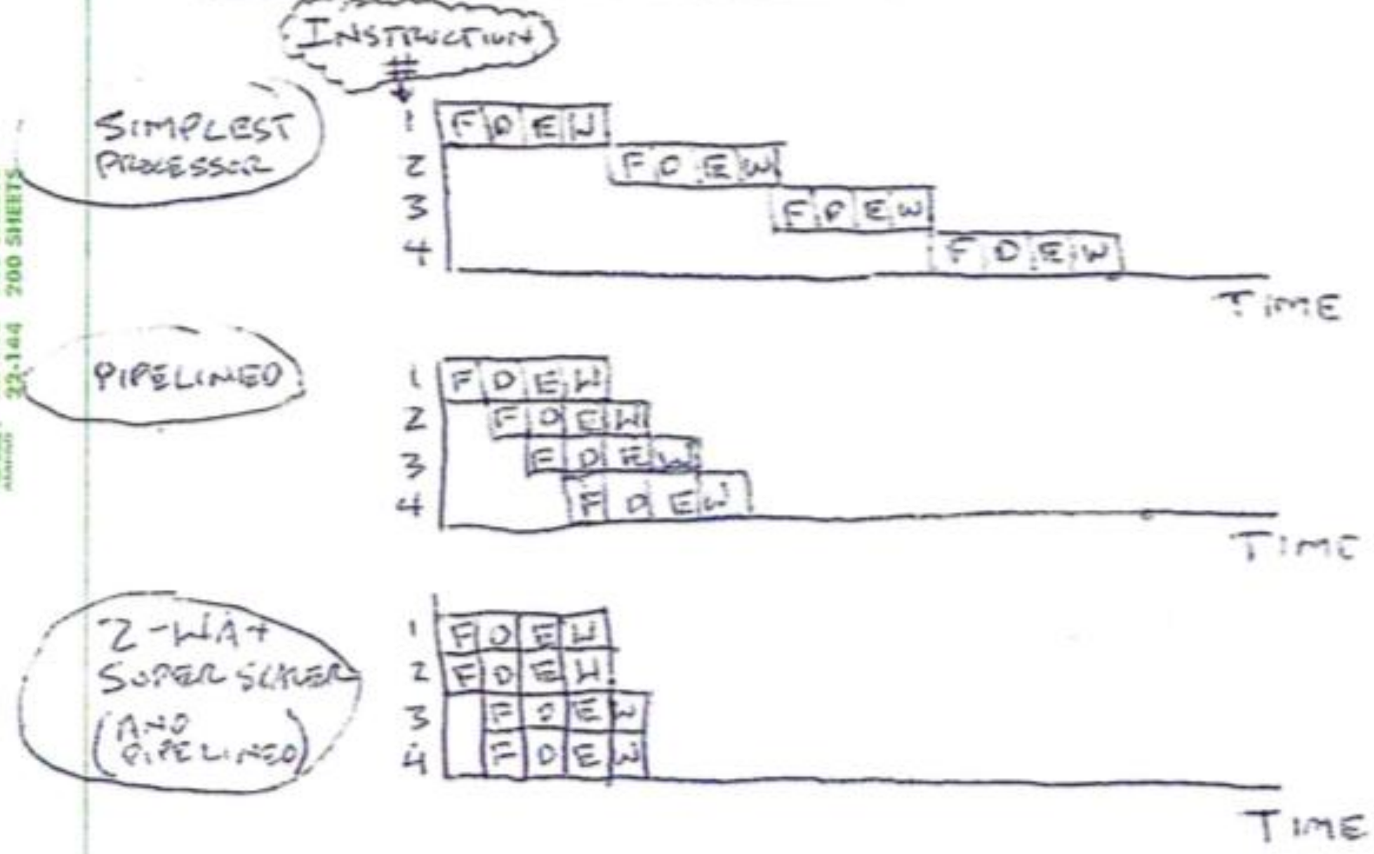
Mnemonic,	operands	Description	Cycles	Status bits affected
bcf	f,b	Clear bit b of register f, where b = 0 to 7	1	
bsf	f,b	Set bit b of register f, where b = 0 to 7	1	
clrw		Clear W	1	Z
clrf	f	Clear f	1	Z
movlw	k	Move literal value to W	1	
movwf	f	Move W to f	1	
movf	f,F(W)	Move f to F or W	1	Z
swapf	f,F(W)	Swap nibbles of f, putting result into F or W	1	
incf	f,F(W)	Increment f, putting result in F or W	1	Z
decf	f,F(W)	Decrement f, putting result in F or W	1	Z
comf	f,F(W)	Complement f, putting result in F or W	1	Z
andlw	k	AND literal value into W	1	Z
andwf	f,F(W)	AND W with f, putting result in F or W	1	Z
iorlw	k	Inclusive-OR literal value into W	1	Z
iorwf	f,F(W)	Inclusive-OR W with f, putting result in F or W	1	Z
xorlw	k	Exclusive-OR literal value into W	1	Z
xorwf	f,F(W)	Exclusive-OR W with f, putting result in F or W	1	Z
addlw	k	Add literal value into W	1	C,DC,Z
addwf	f,F(W)	Add w and f, putting result in F or W	1	C,DC,Z
sublw	k	Subtract W from literal value, putting result in W	1	C,DC,Z
subwf	f,F(W)	Subtract W from f, putting result in F or W	1	C,DC,Z
rlf	f,F(W)	Copy f into F or W; rotate F or W left through the carry bit	1	C
rrf	f,F(W)	Copy f into F or W; rotate F or W right through the carry bit	1	C
btfsc	f,b	Test bit b of register f, where b = 0 to 7; skip if clear	1(2)	
btfss	f,b	Test bit b of register f, where b = 0 to 7; skip if set	1(2)	
decfsz	f,F(W)	Decrement f, putting result in F or W, skip if zero	1(2)	
incfsz	f,F(W)	Increment f, putting result in F or W, skip if zero	1(2)	
goto	label	Go to labeled instruction	2	
call	label	Call labeled subroutine	2	
return		Return from subroutine	2	
retlw	k	Return from subroutine, putting literal value in W	2	
retfie		Return from interrupt service routine; reenable interrupts	2	
clrwdt		Clear watchdog timer	1	NOT_TO,NOT_PD
sleep		Go into standby mode	1	NOT_TO,NOT_PD
nop		No operation	1	

Figure 2-10 PIC16Cxx instruction set.

A <u>VERY</u> SIMPLE MICROCONTROLLER

MACHINE INSTRUCTION TIMING

FETCH (F), DECODE (D), EXECUTE (E), WRITEBACK (W)



3. Amhdahl’s Law for SPEEDUP
    - Overcoming Amhdal’s Law

Amdahl’s Law

by J. Wunderlich, Ph.D.

Amdahl’s Law

The Computer Engineering version of the “law of diminishing returns” or “law of diminishing marginal utility”
Simply put, you can “**speed-up**” computer performance in a measured way; however you get less and less benifit for your effort as you increase your effort.

Amdahl’s Law

“Speed-up” = T_{old} / T_{new}

WHERE T_{old} = time to execute code **<u>prior to</u>** the implementation of a “new feature” to speed-up machine performance

AND T_{new} = time to execute code **<u>after</u>** the implementation of an “new feature” to speed-up machine performance

ALSO, $T_{new} = T_{benifit} + T_{other}$

WHERE $T_{benifit}$ = the **<u>new</u>** time to execute the part of the code that benifits from the “new feature”

AND T_{other} = the time to execute the part of the code that does not benifit from the “new feature”

Amdahl’s Law (example):

Suppose a computer has a code segment that takes 100 msec to execute. And a proposed new ALU could increase the performance of 40% of that code by 10 times. What is the potential speed-up of the entire computer?

T_{old} = 100msec

$T_{benifit}$ = ((40% * (100msec) / 10) = 4msec

T_{other} = (60% * (100msec)) = 60msec

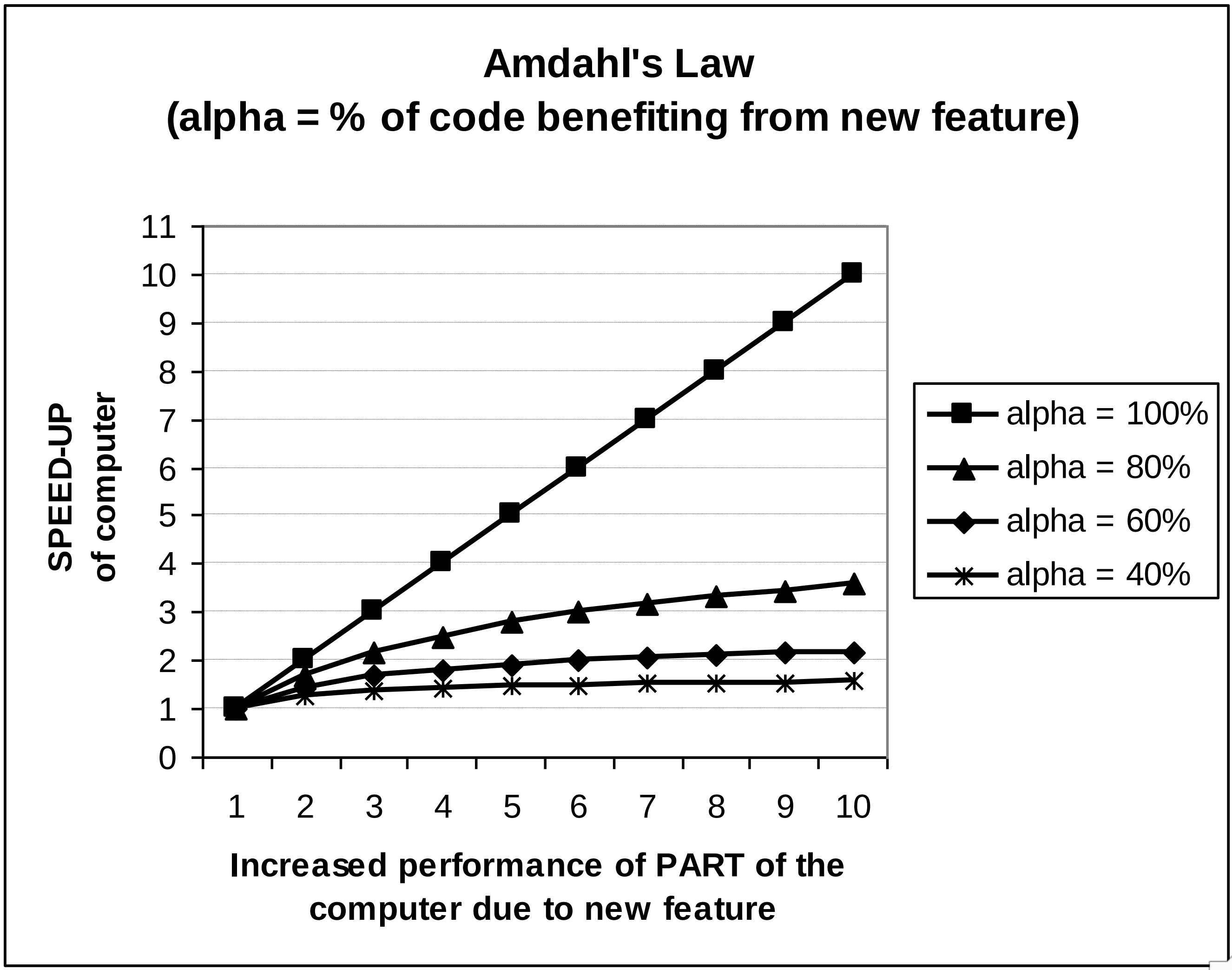
therefore $T_{new} = T_{benifit} + T_{other}$

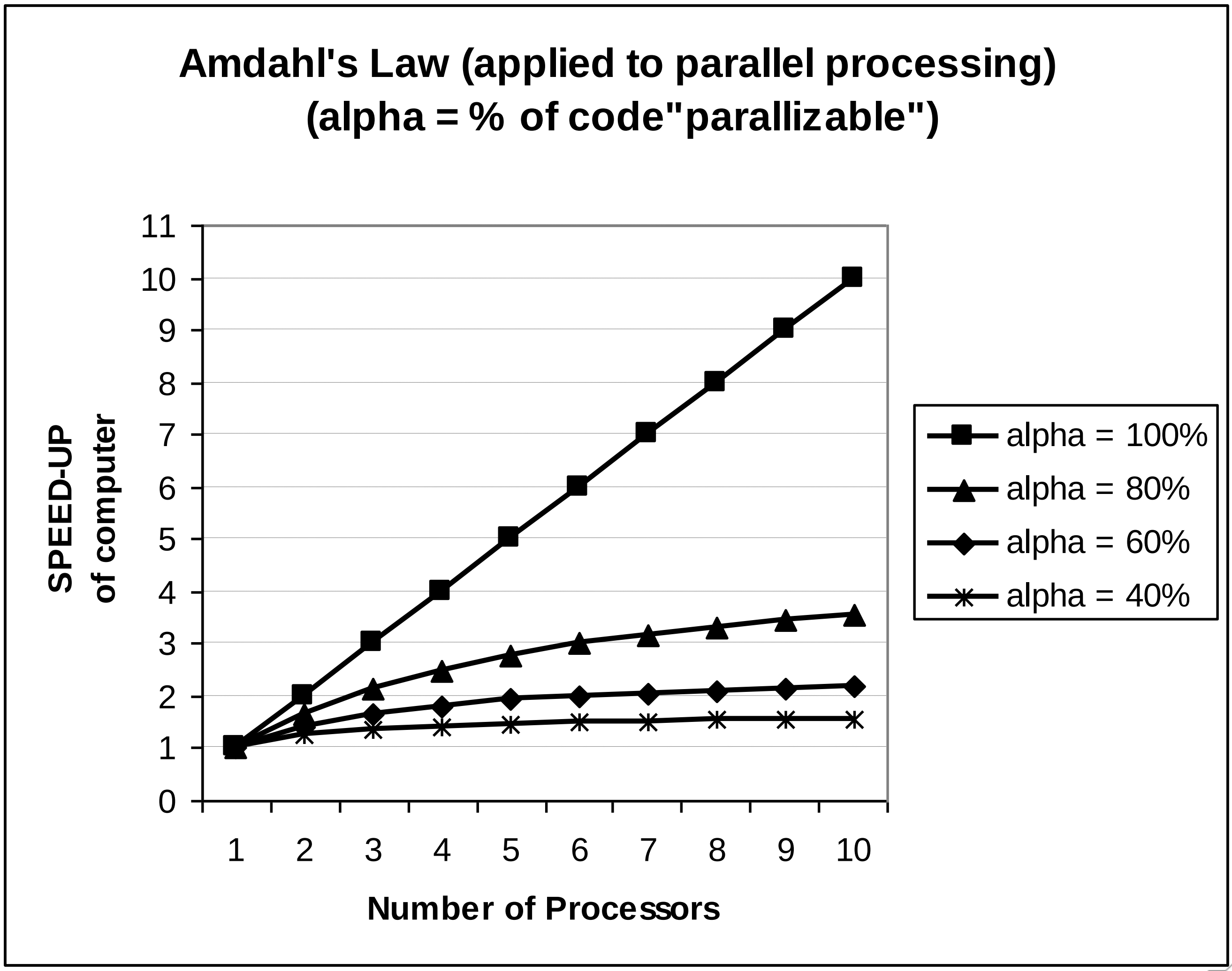
= 4msec + 60msec

= 64msec

“Speed-up” = T_{old} / T_{new}

= 100msec / 64msec = **1.56**


Amdahl's Law
(alpha = % of code benefiting from new feature)
SPEED-UP of computer
Increased performance of PART of the computer due to new feature
alpha = 100%
alpha = 80%
alpha = 60%
alpha = 40%
0
1
2
3
4
5
6
7
8
9
10
11


Amdahl's Law (applied to parallel processing)
(alpha = % of code"parallizable")
SPEED-UP of computer
0
1
2
3
4
5
6
7
8
9
10
11
1
2
3
4
5
6
7
8
9
10
Number of Processors
alpha = 100%
alpha = 80%
alpha = 60%
alpha = 40%

BREAKING THE MULTICORE BOTTLENECK

Simple hardware speeds core-to-core communication

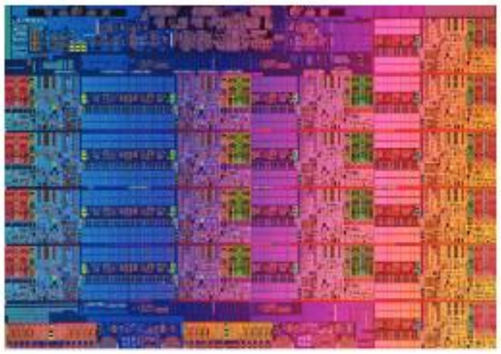
D **Engineers at North** Carolina State University and at Intel have come up with a solution to one of the modern microprocessor's most persistent problems: communication among the processor's many cores. Their answer is a dedicated set of logic circuits they call the Queue Management Device, or QMD. In simulations, integratingthe QMD with the processor's on-chip network at a minimum doubled core-to core communication speed and, in some cases, boosted it much further. Even better, **as the number of cores was increased, the speedup became more pronounced.**

In the last decade, microprocessor designers started putting multiple copies of processor cores ona single die as a way to continue the rate of performance improvement computer makers had enjoyed without causing chip-killing hot spots to form on the CPU. But that solution comes with complications. For one, it means that software programs have to be written so that work is divided among processor cores. The result: Sometimes different cores need to work on the same data or must coordinate the passing of datafrom one core to another.

To prevent the cores from wantonly overwriting one another's information, processing data out of order, or committing other errors, multicore processors use lock-protected software queues. These are data structures that coordinate the movement of and access to information according to software-defined rules. But all that extra software comes with significant overhead, which only gets worse as the number of cores increases. "Communications between cores is becoming a bottleneck," says Yan Solihin, a professor of electrical and computer engineering who led the work at NC State, in Raleigh.

Thesolution-born of a discussion with Intel engineers and executed by Solihin's student, Yipeng Wang, at NC State and at Intel-was to turn the software queue into hardware. This effectively turned three multistep software-queue operations into three simple instructions: Add data to the queue, take data from the queue, and put data close to where it's going to be needed next. Compared with just using the software solution, the QMD sped up a sample task such as

I"



IT'SGETTING CROWDED: This Intel Haswell EXXeon E7 V3 processor has 18 cores trying to· work together without messing up one another's calculations. A bit of additional hardware could speed up communication among the cores.

packet processing - **like network nodes do on the Internet-by a greater and greater amount the more cores were involved.** For16 cores, QMD worked 20 times as fast as the software could.

i"

Once they achieved this result, the engineers reasoned that the QMD might be able to do a few other tricks-such as turning more software into hardware. They added more logic to the QMD and found it could speed up several other core-communications-dependent func tions, including MapReduce, a technol ogy Google pioneered for distributing work to different cores and collecting the results.

Srini Devadas, an expert in cache control systems at MIT, says the QMD addresses "a very important problem." Devadas's own solution for the use of caches by multiple cores-or even mul tiple processors-is more radical than the QMD. Called Tardis, it's a complete rewrite of the cache management rules, and so it is a solution aimed at proces sors and systems of processors further in the future. But QMD, Devadas says, has nearer-term potential. "It's the kind of work that would motivate Intel-putting in a small piece of hardware for a sig nificant improvement."

The Intel engineers involved couldn't comment on whether QMD would find its way into future processors. How ever, they are actively researching its potential. (Wang is now a research sci entist at Intel.) The engineers hope that QMD, among other extensions of the concept, can simplify communication among the cores and the CPU's input/ output system.

Solihin, meanwhile, is inventing other types of hardware accelerators. "We have to improve performance by improving energy efficiency. The only way to do that is to move some software to hardware. The challenge is to figure out which software is used frequently enough that we could justify implement ing it in hardware," he says. "There is a sweet spot." -SAMUEL K. MOORE

This is NOT better than the Ideal Case but *may* be an improvement on the upper bound of Amdahl's Law

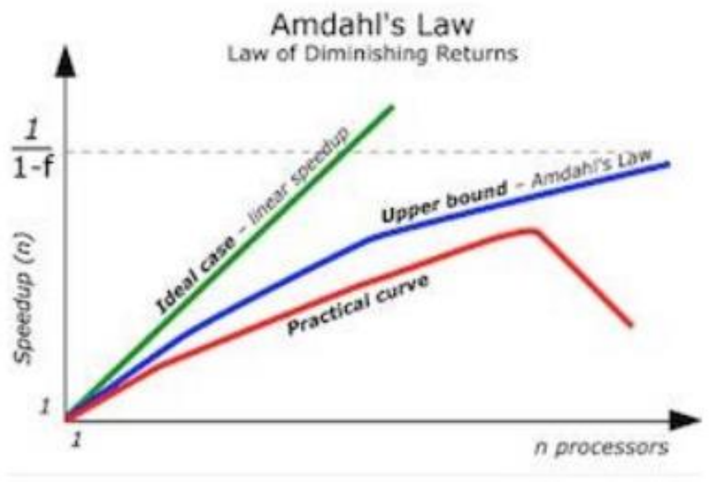
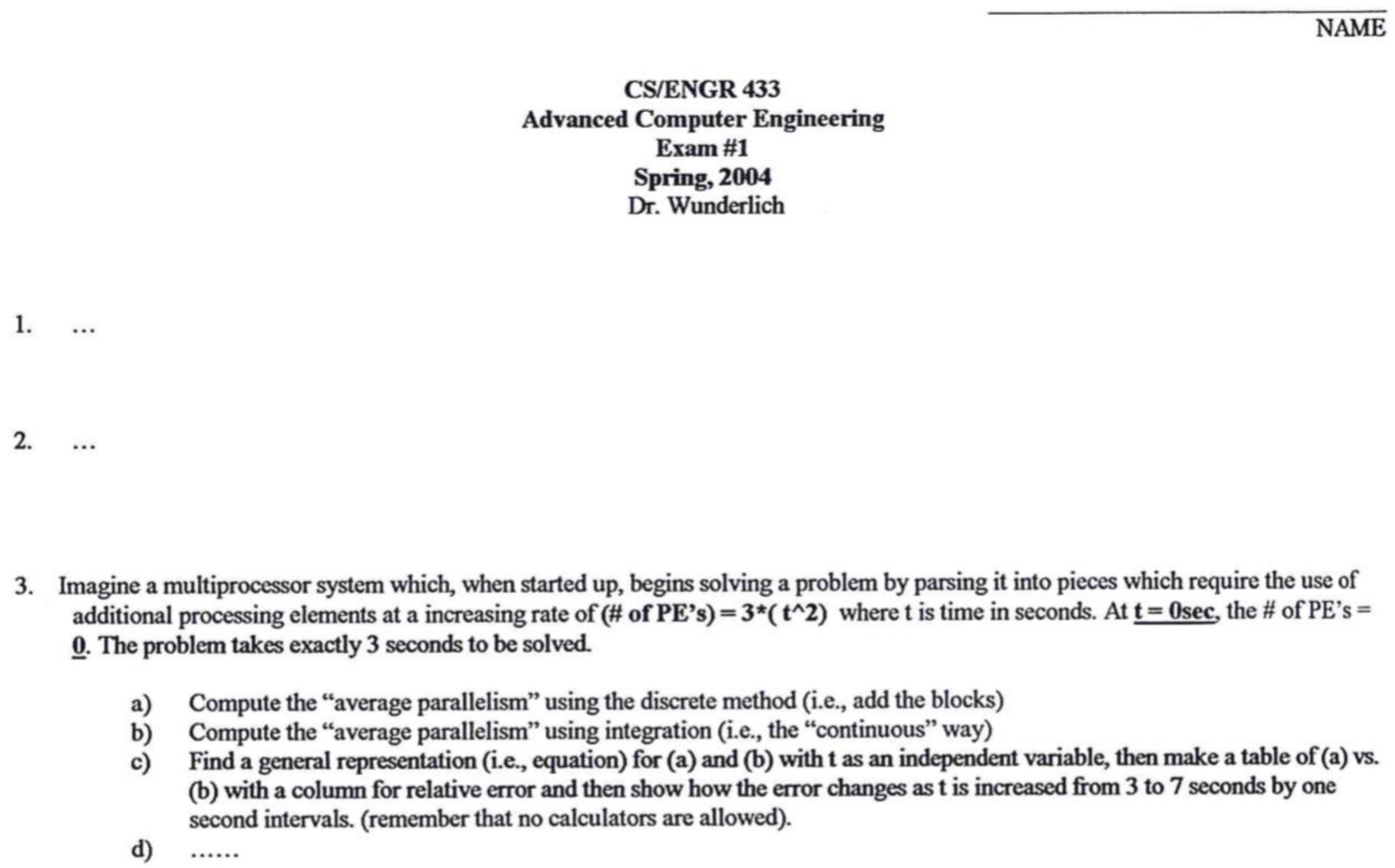


Image from: https://www.javacodegeeks.com/wp-content/uploads/2

.

4. Mathematical Models of Speedup



NAME

CS/ENGR 433
Advanced Computer Engineering
Exam #1
Spring, 2004
Dr. Wunderlich

1. …

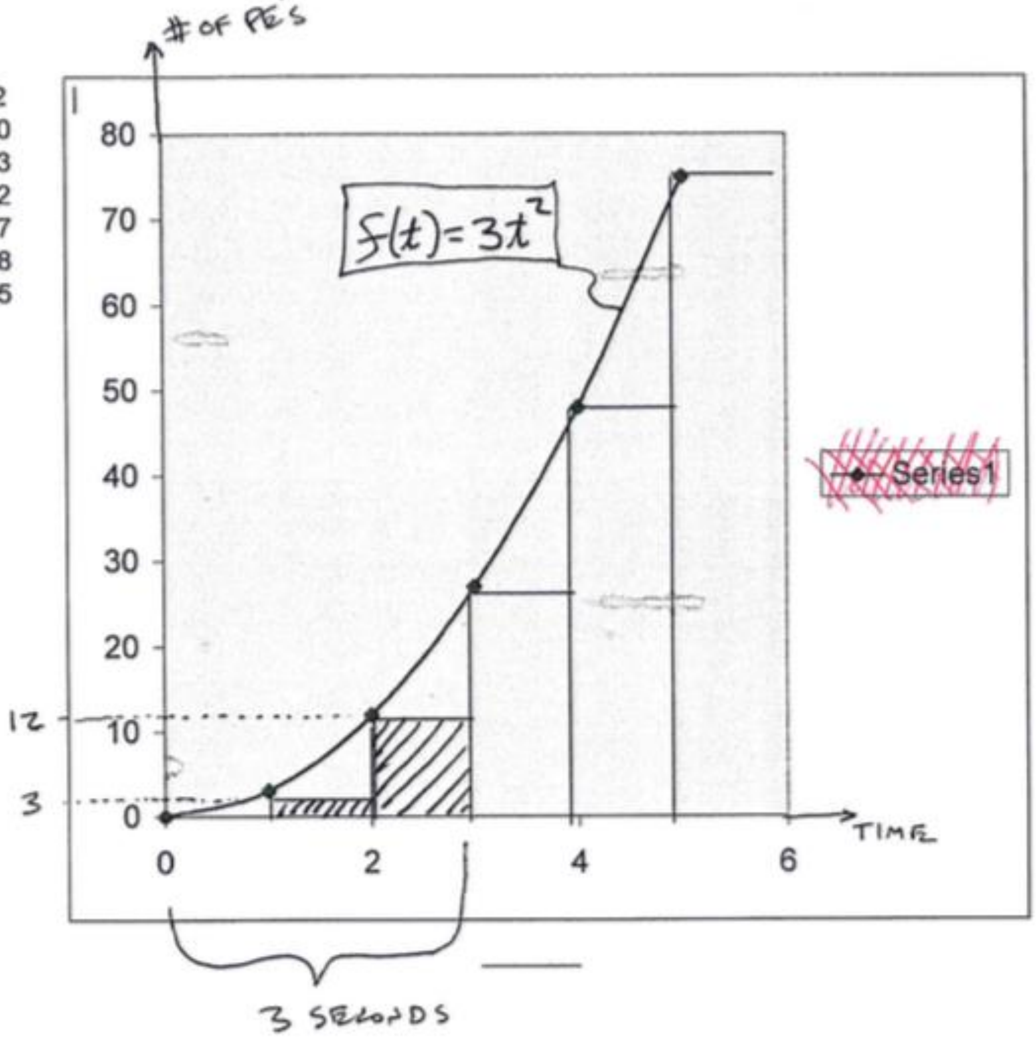
2. …

3. Imagine a multiprocessor system which, when started up, begins solving a problem by parsing it into pieces which require the use of additional processing elements at a increasing rate of **(# of PE's) = 3*(t^2)** where t is time in seconds. At **t = 0sec**, the # of PE's = **0**. The problem takes exactly 3 seconds to be solved.

   a) Compute the "average parallelism" using the discrete method (i.e., add the blocks)
   b) Compute the "average parallelism" using integration (i.e., the "continuous" way)
   c) Find a general representation (i.e., equation) for (a) and (b) with t as an independent variable, then make a table of (a) vs. (b) with a column for relative error and then show how the error changes as t is increased from 3 to 7 seconds by one second intervals. (remember that no calculators are allowed).
   d) ……

CS/ENGR 433 MIDTERM#1 PROBLEM#3 SOLUTION

t	3t^2
0	0
1	3
2	12
3	27
4	48
5	75



A) DISCRETE:

$$A = \left(\sum_{i=1}^{m} i \cdot t_i\right) \Big/ \left(\sum_{i=1}^{m} t_i\right) = \frac{(0*1)+(3*1)+(12*1)}{1+1+1} = \frac{15}{3} = \boxed{5}$$

B) CONTINUOUS:

$$A = \frac{1}{t_{upper} - t_{lower}} \int_{t_{lower}}^{t_{upper}} f(t)\,dt = \frac{1}{3-0}\left(3\ \frac{t^3}{3}\Big|_{t_{lower}}^{t_{upper}}\right) = \frac{1}{3}\left(3^3 - 0^3\right) = \frac{27}{3} = \boxed{9}$$

C) GENERAL FORMS:

DISCRETE

$$A = \left(\frac{f(0)*1 + f(1)*1 + f(2)*1}{1+1+1}\right) \text{ FOR } T=3$$

IN GENERAL: $A = \left(\frac{f(0) + f(1) + \dots + f(T-1)}{T}\right)$

AND SINCE $f(\,)=3(\,)^2$, $A = \left(\frac{3(0)^2 + 3(T-2)^2 + 3(T-1)^2}{T}\right)$

$$A = \frac{3}{T}\left[\sum_{n=1}^{T-1} n^2\right]$$

CONTINUOUS

$$A = \frac{1}{T}\left(T^3 - 0^3\right)$$

$$A = \frac{T^3}{T}$$

$$A = T^2$$

CONTINUED....

T	DISCRETE $A_D = \frac{3}{T}\left[\sum_{n=1}^{T-1} n^2\right]$	CONTINUOUS $A_C = T^2$	RELATIVE ERROR (%) $E = \left(\frac{A_C - A_D}{A_D}\right) \times 100$
3	$\frac{3}{3}\left[1^2+2^2\right] = 5$	9	$E = \left(\frac{9-5}{5}\right)\times 100 = \left(\frac{4}{5}\right)\times 100 = \boxed{80\%}$
4	$\frac{3}{4}\left[1+4+9\right] = \frac{3}{4}\left[14\right] = \frac{21}{2}$	16	$E = \left(\frac{16-21/2}{21/2}\right)\times 100 = \left(\frac{11}{21}\right)\times 100 \approx \boxed{50\%}$
5	$\frac{3}{5}\left[1+4+9+16\right] = \frac{3}{5}\left[30\right] = 18$	25	$E = \left(\frac{25-18}{18}\right)\times 100 = \left(\frac{7}{18}\right)\times 100 \approx \boxed{40\%}$
6	27.5	36	$\approx \boxed{31\%}$
7	39	49	$\approx \boxed{26\%}$

7.2

The Area under a Curve

Let us now consider the problem of finding the area of a given plane region. We have an intuitive idea of what we mean by the area of a plane region; yet when someone says that the area of some irregular region is 5 square units, exactly what is meant? In other words, can we give a general definition of the area of a plane region? Before attempting to do so, let us consider some properties of area.

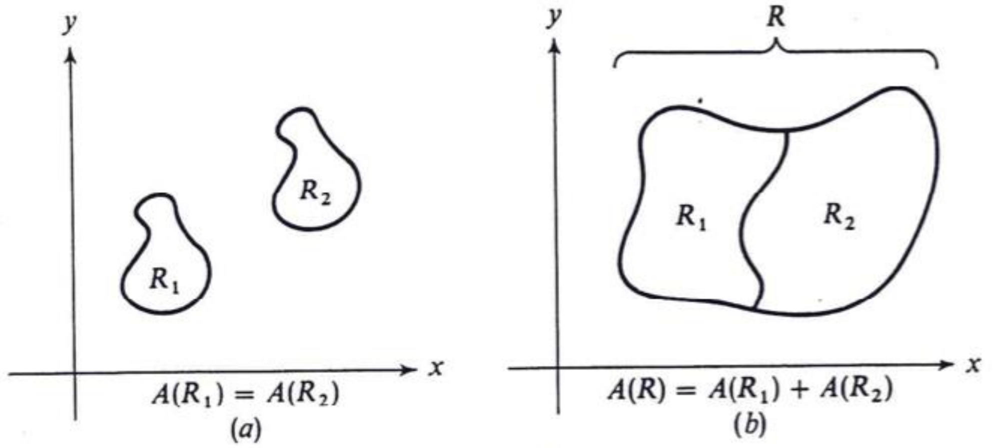


Figure 7.1

1. If R is any region, then the area of R, $A(R)$, is a real number and $A(R) \geq 0$.
2. If R_1 and R_2 are congruent regions, then $A(R_1) = A(R_2)$ (see Figure 7.1(a)).
3. If $R = R_1 \cup R_2$, where R_1 and R_2 have only boundary points in common, then $A(R) = A(R_1) + A(R_2)$ (see Figure 7.1(b)).

While these three properties tell us something about area, they give us no way of assigning a specific number as the area of a given region. In order to make such an assignment, we consider the area of a very simple region.

4. The area of a rectangle of length l and width w is $A = lw$.

With these four properties it is possible to determine the area of any polygon (recall that a polygon has a finite number of *straight* sides). But suppose we want to find the area "under the curve" $y = x^2$ from $x = 0$ to $x = 1$, that is, the area of the region bounded by $y = x^2$, the x axis, and the vertical lines $x = 0$ and $x = 1$ (see Figure 7.2). This region has one curved side. How can we handle this case? Suppose we take a lesson from our discussion of the slope of a graph. There we approximated the tangent line with a secant line and noted what happened to its slope as it moved closer and closer to the tangent line

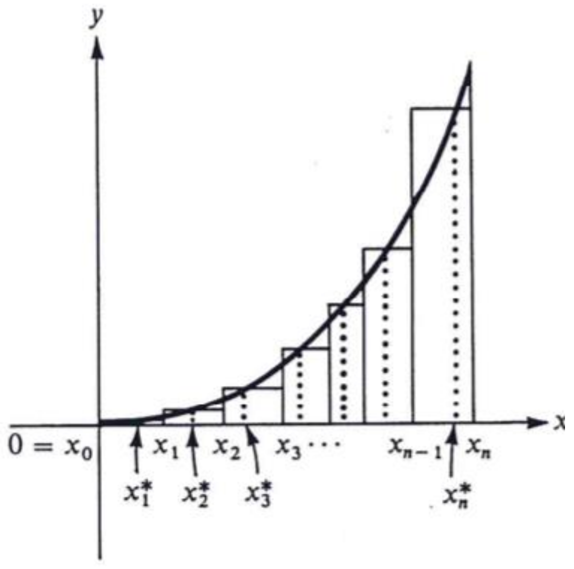


Figure 7.2

Since the area of a rectangle is known, let us approximate the area we want with areas of rectangles. This is done in the following way. We subdivide the interval [0, 1] into n (not necessarily equal) subintervals (see Figure 7.2) by means of the numbers

$$x_0, x_1, x_2, \ldots, x_{n-1}, x_n, \quad \text{where} \quad x_0 = 0 \text{ and } x_n = 1.$$

Within each subinterval we select a number in any way we choose from the left- to the right-hand end point. Let us call these numbers

$$x_1^*, x_2^*, x_3^*, \ldots, x_n^*.$$

Now let us construct a rectangle for each subinterval, using the subinterval itself as the base and $f(x_i^*)$ as the altitude. Thus the sum of the areas of all of these rectangles gives an approximation (although perhaps a very poor one) of the area we seek. This sum is

$$f(x_1^*)(x_1 - x_0) + f(x_2^*)(x_2 - x_1) + f(x_3^*)(x_3 - x_2) + \cdots + f(x_n^*)(x_n - x_{n-1}) = \sum_{i=1}^{n} f(x_i^*)(x_i - x_{i-1}).$$

In order to simplify this expression, we reintroduce the delta notation that was used in Section 6.3. It was noted there that Δx represents a difference of two values of x. In particular, we define

$$\Delta x_i = x_i - x_{i-1}.$$

Thus Δx_i represents the width of the ith rectangle. With this notation our approximating sum is

$$\sum_{i=1}^{n} f(x_i^*)\, \Delta x_i.$$

Let us now consider the question of the error committed in making this approximation. There are two types of errors (see Figure 7.3), which we shall call positive and negative errors. A positive error occurs when a portion of a rectangle lies outside the original region R; errors of this type tend to make the sum greater than the area desired. A negative error occurs when a portion of R lies outside all of the rectangles; errors of this type tend to make the sum less than the area desired. In order to get a better approximation of the area of R, we must find a way to decrease both types of errors. Let us consider the positive errors first.

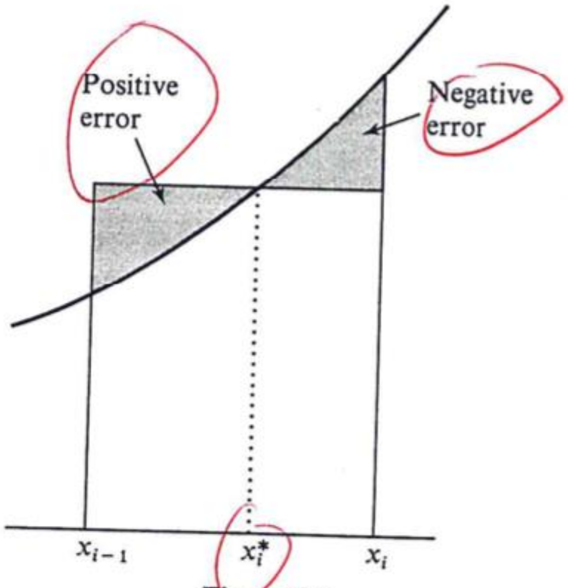


Figure 7.3

Suppose the interval [0, 1] is subdivided as shown in Figure 7.4(a). If the x_i^*'s are chosen to be the right-hand end points, the positive error is the largest possible one for that subdivision. It is represented by the shaded portion of Figure 7.4(a). Now suppose that each interval of the first subdivision is itself subdivided to give the finer subdivision of Figure 7.4(b). The largest possible positive error is now smaller than it was for the first subdivision. This error is represented by the shaded portion of Figure 7.4(b). The difference between this error and the first is represented by the unshaded rectangles above the curve. By further subdivision, as in Figure 7.4(c), the largest possible positive error is reduced still further. Thus, as we take ever finer subdivisions, the maximum positive error (and thus any positive error) can be made as small as we choose. A similar analysis shows that we can make the negative error as small as we choose by taking the subdivision fine enough.

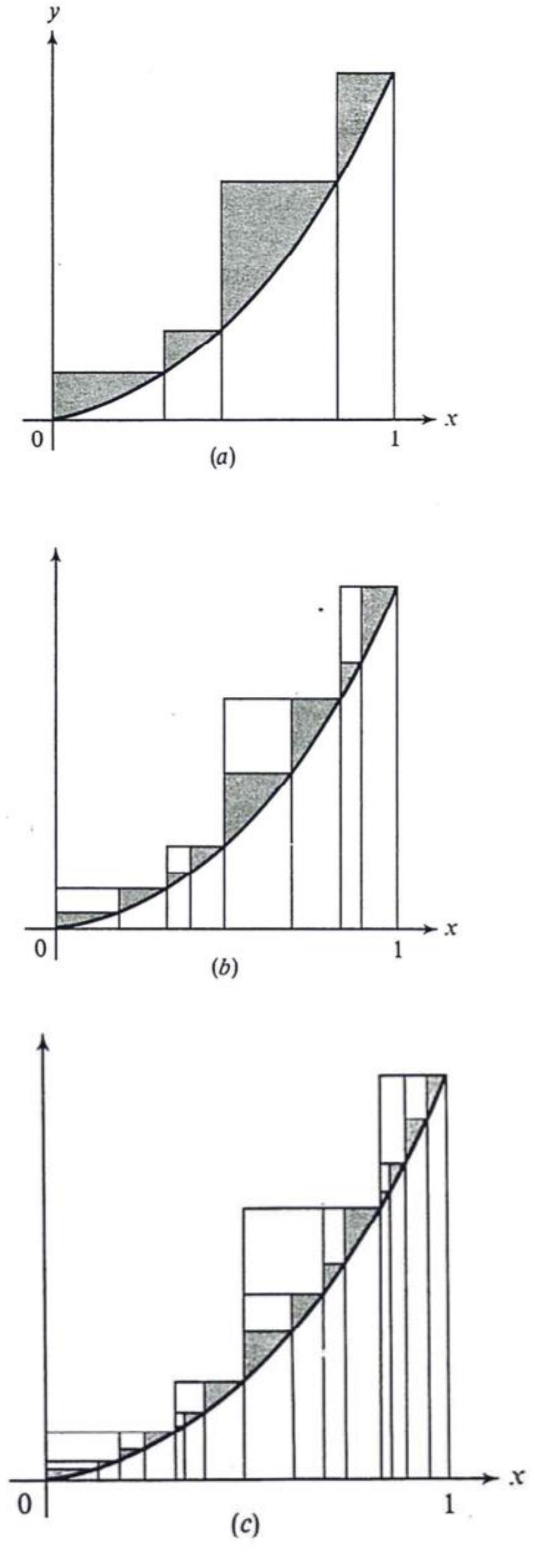


Figure 7.4

All of the foregoing analysis suggests a limiting process. Thus it appears that the area we want is the limit of the approximating sum as the lengths of the subintervals approach zero. The analysis also suggests that the result is independent of the way in which we subdivide (as long as the lengths of all of the subintervals approach zero) and the way in which the x_i^*'s are chosen. Let us now apply this method to our original problem.

Example 1

Find the area under the curve $y = x^2$ from $x = 0$ to $x = 1$.

First of all, we must subdivide the interval $[0, 1]$ and then choose the x_i^*'s. In order to simplify the algebra involved, let us subdivide the interval into *n equal* subintervals and choose the x_i^*'s to be the right-hand end points of the subintervals (see Figure 7.5). Thus

$$x_0 = \frac{0}{n} = 0, \qquad x_1 = \frac{1}{n}, \qquad x_2 = \frac{2}{n}, \qquad \vdots \qquad x_i = \frac{i}{n}, \qquad \vdots \qquad x_n = \frac{n}{n} = 1.$$

$$x_1^* = \frac{1}{n}, \qquad x_2^* = \frac{2}{n}, \qquad \vdots \qquad x_i^* = \frac{i}{n}, \qquad \vdots \qquad x_n^* = \frac{n}{n} = 1,$$

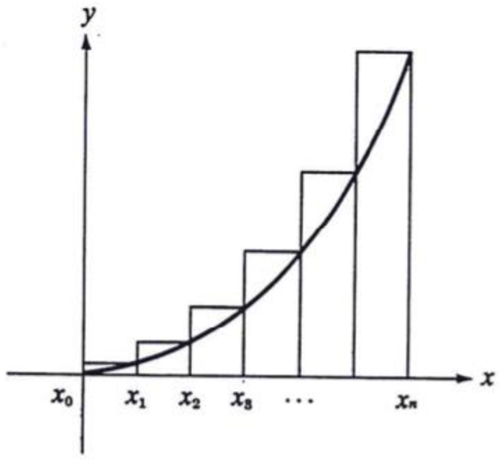


Figure 7.5

We see that the length of each of the n equal subintervals is $1/n$, and the lengths of all of them are approaching zero as n gets large and positive. This last observation allows us to simplify our notation for the limit.

$$\begin{aligned}
A &= \lim_{n \to +\infty} \sum_{i=1}^{n} f(x_i^*)\Delta x_i \\
&= \lim_{n \to +\infty} \sum_{i=1}^{n} f\left(\frac{i}{n}\right)\left(\frac{1}{n}\right) \\
&= \lim_{n \to +\infty} \sum_{i=1}^{n} \frac{i^2}{n^2} \cdot \frac{1}{n} && \text{(See Note 1.)} \\
&= \lim_{n \to +\infty} \frac{1}{n^3} \sum_{i=1}^{n} i^2 && \text{(By Theorem 7.3(a))} \\
&= \lim_{n \to +\infty} \frac{1}{n^3} \frac{n(n+1)(2n+1)}{6} && \text{(By Theorem 7.2(c))} \\
&= \lim_{n \to +\infty} \frac{\left(1 + \frac{1}{n}\right)\left(2 + \frac{1}{n}\right)}{6} && \text{(See Note 2.)} \\
&= \frac{1}{3}.
\end{aligned}$$

Note 1: Since our original function is in the form $f(x) = x^2$, it follows that

$$f(i/n) = (i/n)^2 = i^2/n^2.$$

Note 2: We have divided both numerator and denominator by n^3. In the numerator, each factor was divided by one of the n's.

Perhaps you feel critical of our answer in Example 1, because we used the right-hand end points throughout, so that our approximating sums are *all* bigger than the area we want. If the answer is incorrect, it must be too large. Let us now use the left-hand end points. Since the approximating sums are all smaller than the area we want (see Figure 7.6), we feel that the result will certainly not be greater than the area under the curve —either it will be the area we want, or it will be less than that area. In this case $x_i = i/n$ as before, but $x_i^* = (i-1)/n$. Thus

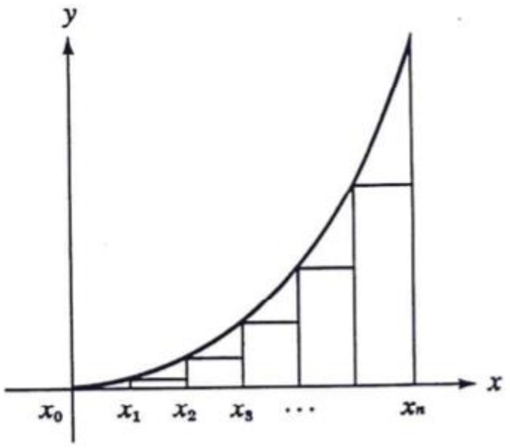


Figure 7.6

$$\begin{aligned}
A &= \lim_{n\to+\infty} \sum_{i=1}^{n} f(x_i^*)\Delta x_i \\
&= \lim_{n\to+\infty} \sum_{i=1}^{n} f\left(\frac{i-1}{n}\right)\left(\frac{1}{n}\right) \\
&= \lim_{n\to+\infty} \sum_{i=1}^{n} \frac{(i-1)^2}{n^2}\cdot\frac{1}{n} \\
&= \lim_{n\to+\infty} \frac{1}{n^3}\sum_{i=1}^{n} (i-1)^2 \\
&= \lim_{n\to+\infty} \frac{1}{n^3}\frac{(n-1)n(2n-1)}{6} \qquad \text{(see Problem 33, Section 7.1)} \\
&= \lim_{n\to+\infty} \frac{\left(1-\frac{1}{n}\right)\left(2-\frac{1}{n}\right)}{6} \\
&= \frac{1}{3}.
\end{aligned}$$

We see that we have exactly the same result as in the previous case. It seems reasonable to expect that, if we get 1/3 in both of these extreme cases, we should get 1/3 in any case.

The fact that the result is independent of the choice of the x_i^*'s is important. Similarly, it is important to note that the result is independent of the choice of the subdivision (see Problem 20). Unfortunately, these important facts are also difficult to prove, and we shall not attempt proofs here.

Example 2

Find the area under the curve $y = x^2$, from $x = 1$ to $x = 3$.

Since we have inferred that the result we get is independent of both the subdivision and the choice of the x_i^*'s, we shall subdivide the interval [1, 3] into n equal intervals and choose the x_i^*'s to be right-hand end points (see Figure 7.7). Because we are subdividing an interval of length 2 into n equal intervals, the length of each is $2/n$. Thus

$$\begin{aligned}
x_0 &= 1, & x_1^* &= 1 + \frac{2}{n},\\
x_1 &= 1 + \frac{2}{n}, & x_2^* &= 1 + \frac{4}{n},\\
x_2 &= 1 + \frac{4}{n}, & &\vdots\\
&\vdots & x_i^* &= 1 + \frac{2i}{n},\\
x_i &= 1 + \frac{2i}{n}, & &\vdots\\
&\vdots & x_n^* &= 1 + \frac{2n}{n},\\
x_n &= 1 + \frac{2n}{n} = 3.
\end{aligned}$$

$$\begin{aligned}
A &= \lim_{n\to+\infty} \sum_{i=1}^{n} f(x_i^*)\Delta x_i\\
&= \lim_{n\to+\infty} \sum_{i=1}^{n} f\left(1 + \frac{2i}{n}\right)\frac{2}{n}\\
&= \lim_{n\to+\infty} \sum_{i=1}^{n} \left(1 + \frac{2i}{n}\right)^2\frac{2}{n}\\
&= \lim_{n\to+\infty} \sum_{i=1}^{n} \left(1 + \frac{4i}{n} + \frac{4i^2}{n^2}\right)\frac{2}{n}\\
&= \lim_{n\to+\infty} \frac{2}{n}\left[n + \frac{4}{n}\frac{n(n+1)}{2} + \frac{4}{n^2}\frac{n(n+1)(2n+1)}{6}\right]\\
&= \lim_{n\to+\infty}\left[2 + 4\left(1 + \frac{1}{n}\right) + \frac{4}{3}\left(1 + \frac{1}{n}\right)\cdot\left(2 + \frac{1}{n}\right)\right] = 2 + 4 + \frac{8}{3} = \frac{26}{3}.
\end{aligned}$$

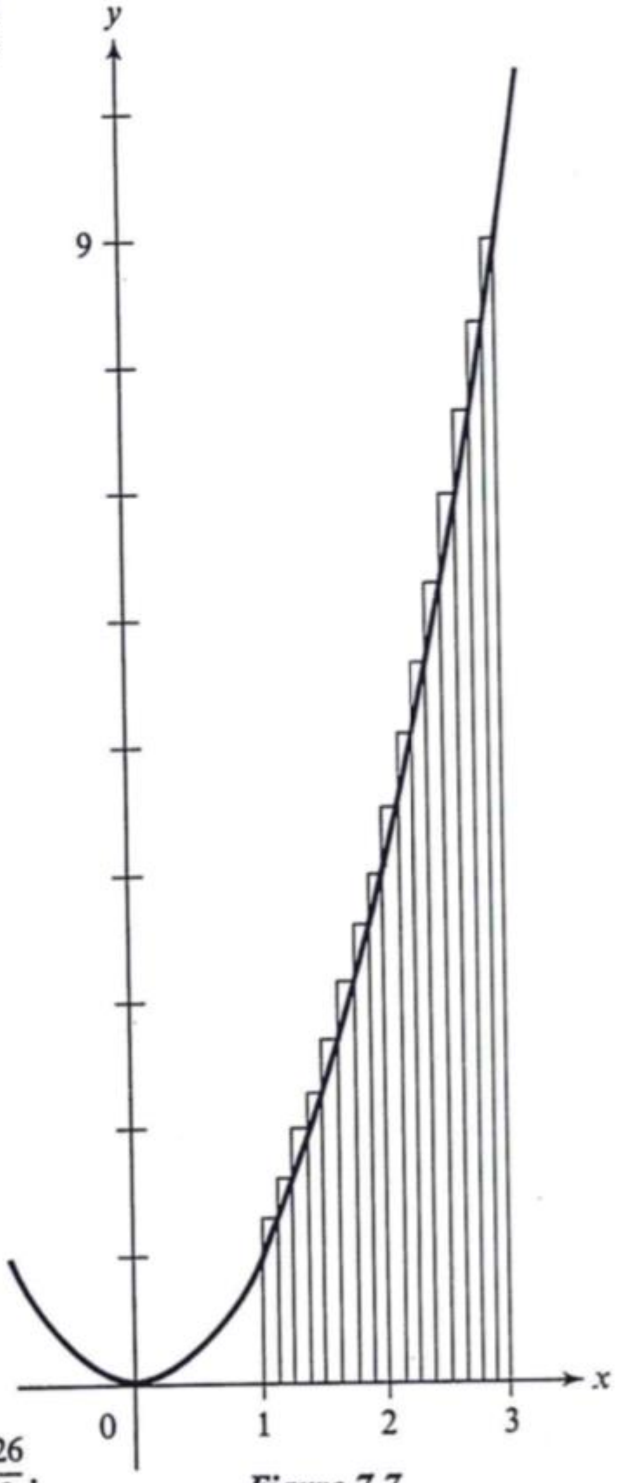
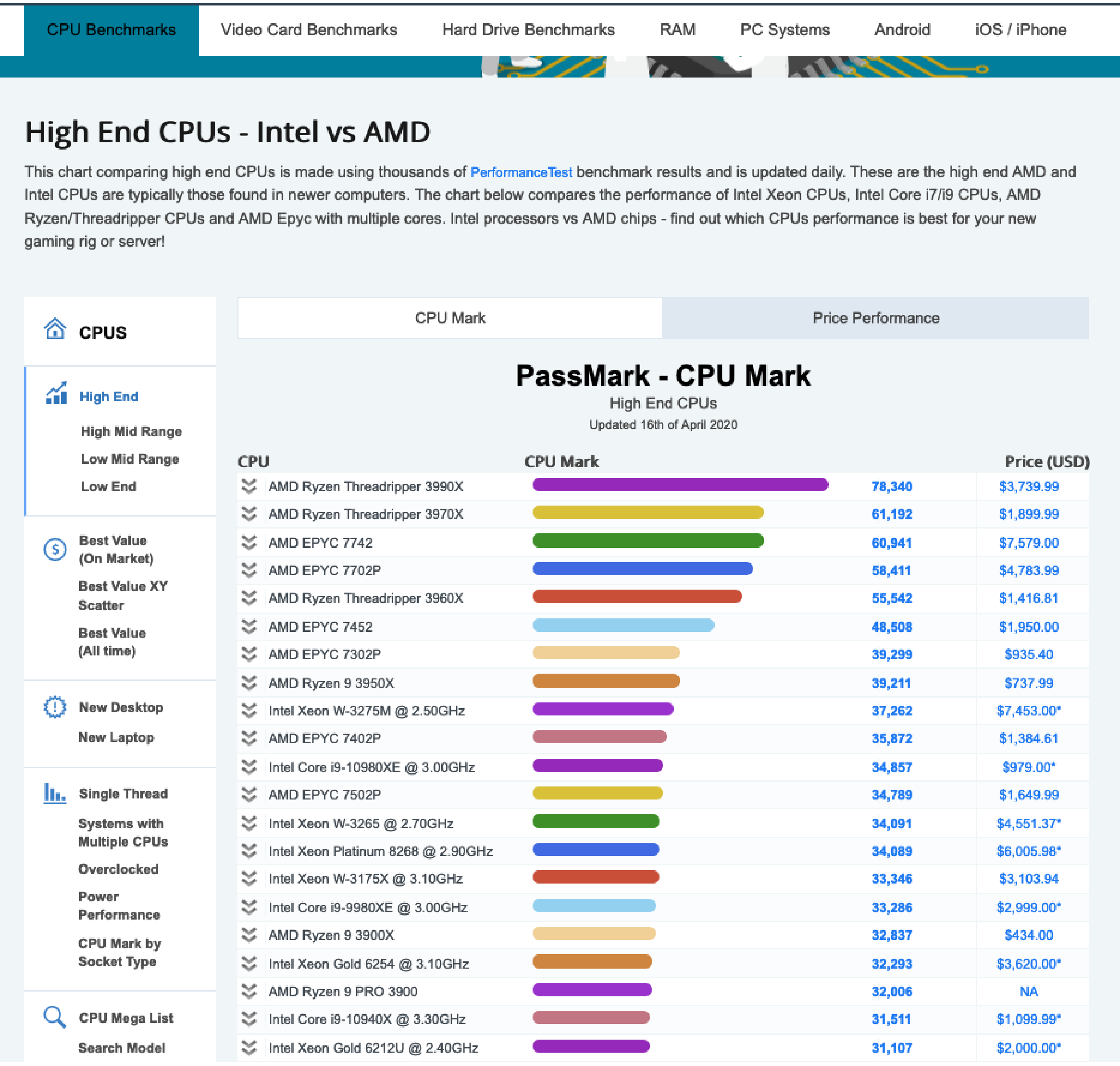


Figure 7.7

5. Common Benchmarks



High End CPUs - Intel vs AMD

This chart comparing high end CPUs is made using thousands of PerformanceTest benchmark results and is updated daily. These are the high end AMD and Intel CPUs are typically those found in newer computers. The chart below compares the performance of Intel Xeon CPUs, Intel Core i7/i9 CPUs, AMD Ryzen/Threadripper CPUs and AMD Epyc with multiple cores. Intel processors vs AMD chips - find out which CPUs performance is best for your new gaming rig or server!

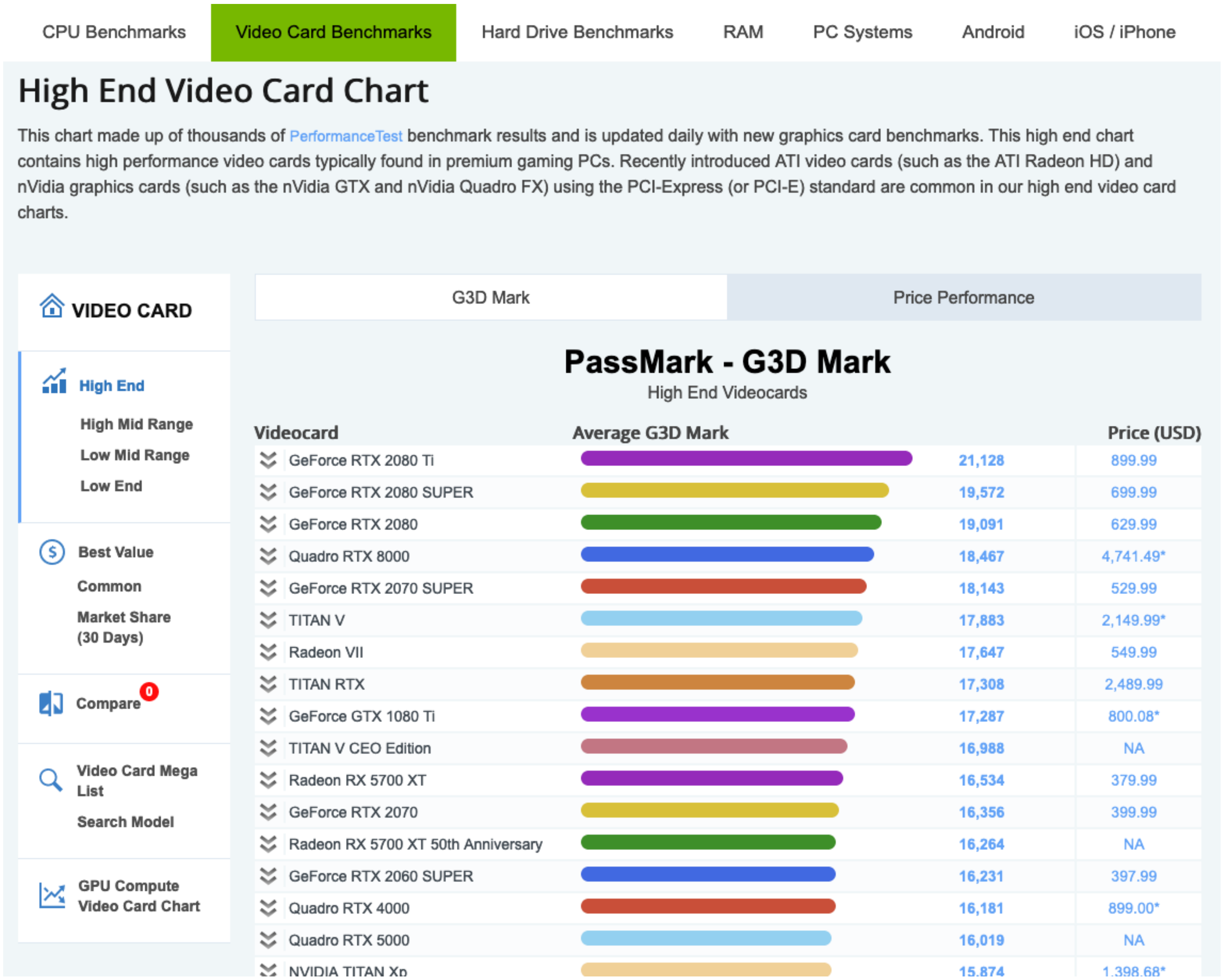
PassMark - CPU Mark

High End CPUs

Updated 16th of April 2020

CPU	CPU Mark	Price (USD)
AMD Ryzen Threadripper 3990X	78,340	$3,739.99
AMD Ryzen Threadripper 3970X	61,192	$1,899.99
AMD EPYC 7742	60,941	$7,579.00
AMD EPYC 7702P	58,411	$4,783.99
AMD Ryzen Threadripper 3960X	55,542	$1,416.81
AMD EPYC 7452	48,508	$1,950.00
AMD EPYC 7302P	39,299	$935.40
AMD Ryzen 9 3950X	39,211	$737.99
Intel Xeon W-3275M @ 2.50GHz	37,262	$7,453.00*
AMD EPYC 7402P	35,872	$1,384.61
Intel Core i9-10980XE @ 3.00GHz	34,857	$979.00*
AMD EPYC 7502P	34,789	$1,649.99
Intel Xeon W-3265 @ 2.70GHz	34,091	$4,551.37*
Intel Xeon Platinum 8268 @ 2.90GHz	34,089	$6,005.98*
Intel Xeon W-3175X @ 3.10GHz	33,346	$3,103.94
Intel Core i9-9980XE @ 3.00GHz	33,286	$2,999.00*
AMD Ryzen 9 3900X	32,837	$434.00
Intel Xeon Gold 6254 @ 3.10GHz	32,293	$3,620.00*
AMD Ryzen 9 PRO 3900	32,006	NA
Intel Core i9-10940X @ 3.30GHz	31,511	$1,099.99*
Intel Xeon Gold 6212U @ 2.40GHz	31,107	$2,000.00*

https://www.cpubenchmark.net/high_end_cpus.html



CPU Benchmarks | Video Card Benchmarks | Hard Drive Benchmarks | RAM | PC Systems | Android | iOS / iPhone

High End Video Card Chart

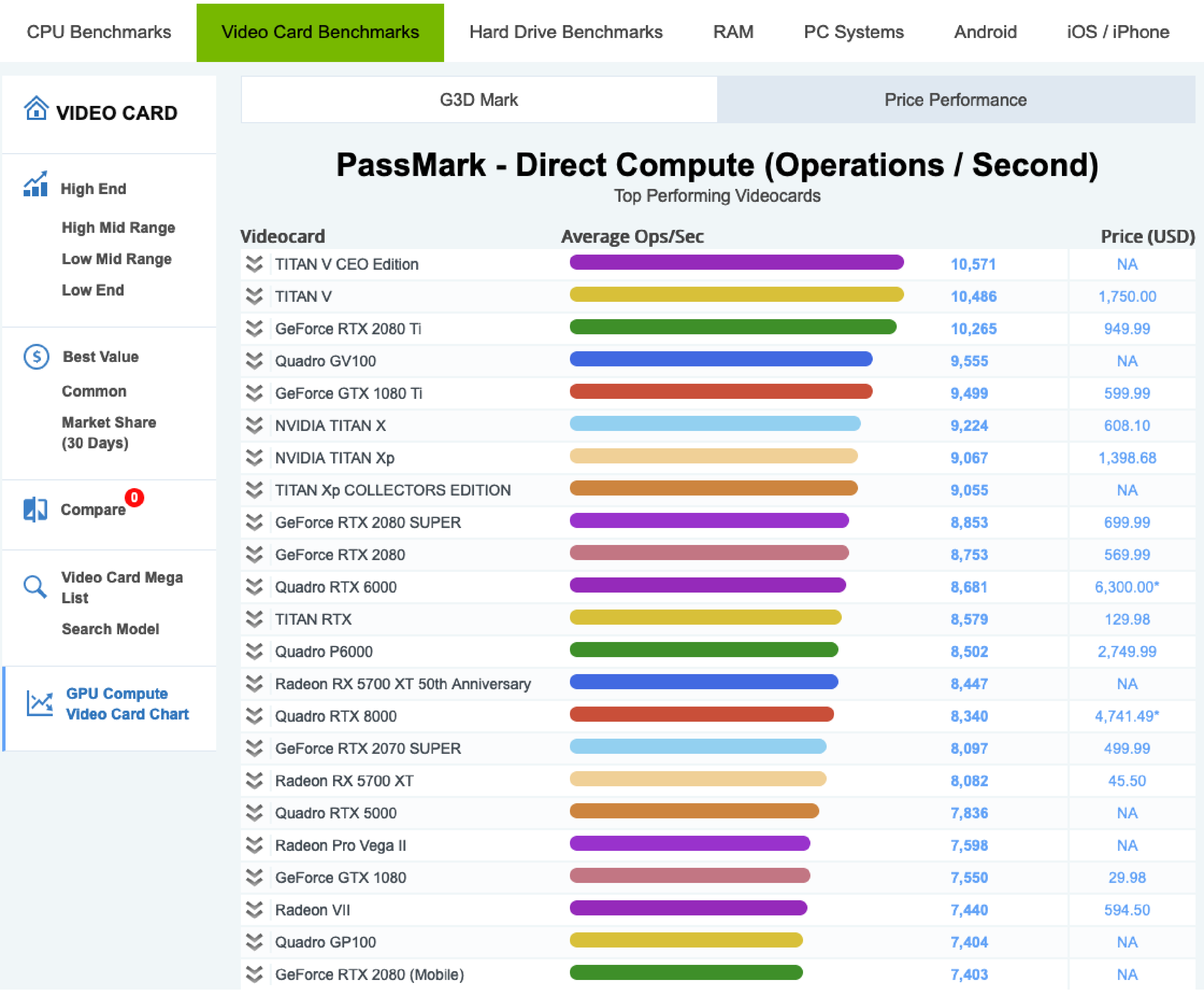
This chart made up of thousands of PerformanceTest benchmark results and is updated daily with new graphics card benchmarks. This high end chart contains high performance video cards typically found in premium gaming PCs. Recently introduced ATI video cards (such as the ATI Radeon HD) and nVidia graphics cards (such as the nVidia GTX and nVidia Quadro FX) using the PCI-Express (or PCI-E) standard are common in our high end video card charts.

PassMark - G3D Mark

High End Videocards

Videocard	Average G3D Mark	Price (USD)
GeForce RTX 2080 Ti	21,128	899.99
GeForce RTX 2080 SUPER	19,572	699.99
GeForce RTX 2080	19,091	629.99
Quadro RTX 8000	18,467	4,741.49*
GeForce RTX 2070 SUPER	18,143	529.99
TITAN V	17,883	2,149.99*
Radeon VII	17,647	549.99
TITAN RTX	17,308	2,489.99
GeForce GTX 1080 Ti	17,287	800.08*
TITAN V CEO Edition	16,988	NA
Radeon RX 5700 XT	16,534	379.99
GeForce RTX 2070	16,356	399.99
Radeon RX 5700 XT 50th Anniversary	16,264	NA
GeForce RTX 2060 SUPER	16,231	397.99
Quadro RTX 4000	16,181	899.00*
Quadro RTX 5000	16,019	NA
NVIDIA TITAN Xp	15,874	1,398.68*

https://www.cpubenchmark.net/high_end_cpus.html


CPU Benchmarks | Video Card Benchmarks | Hard Drive Benchmarks | RAM | PC Systems | Android | iOS / iPhone

VIDEO CARD

High End
High Mid Range
Low Mid Range
Low End

Best Value
Common
Market Share (30 Days)

Compare 0

Video Card Mega List
Search Model

GPU Compute Video Card Chart

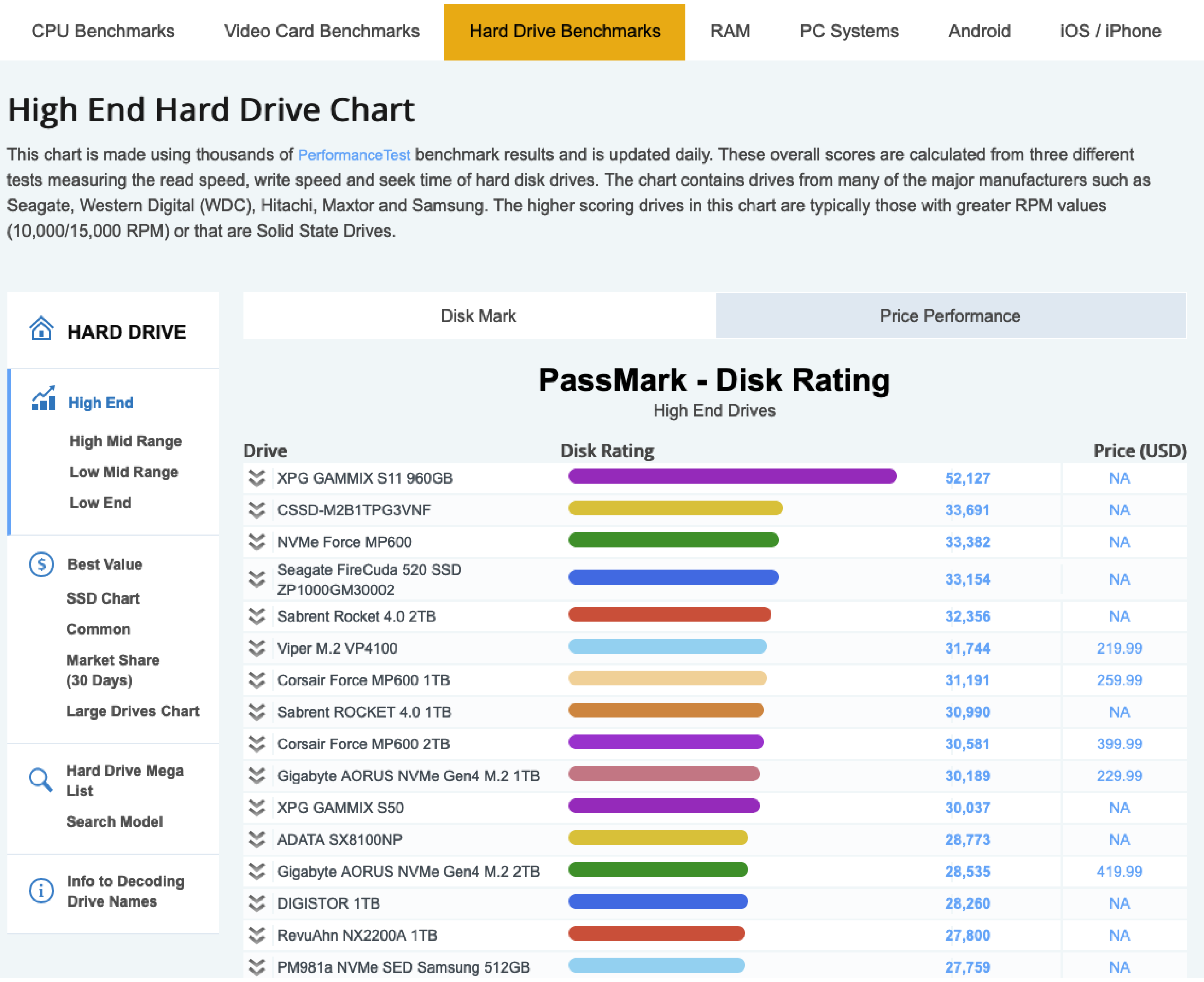
G3D Mark | Price Performance

PassMark - Direct Compute (Operations / Second)

Top Performing Videocards

Videocard	Average Ops/Sec	Price (USD)
TITAN V CEO Edition	10,571	NA
TITAN V	10,486	1,750.00
GeForce RTX 2080 Ti	10,265	949.99
Quadro GV100	9,555	NA
GeForce GTX 1080 Ti	9,499	599.99
NVIDIA TITAN X	9,224	608.10
NVIDIA TITAN Xp	9,067	1,398.68
TITAN Xp COLLECTORS EDITION	9,055	NA
GeForce RTX 2080 SUPER	8,853	699.99
GeForce RTX 2080	8,753	569.99
Quadro RTX 6000	8,681	6,300.00*
TITAN RTX	8,579	129.98
Quadro P6000	8,502	2,749.99
Radeon RX 5700 XT 50th Anniversary	8,447	NA
Quadro RTX 8000	8,340	4,741.49*
GeForce RTX 2070 SUPER	8,097	499.99
Radeon RX 5700 XT	8,082	45.50
Quadro RTX 5000	7,836	NA
Radeon Pro Vega II	7,598	NA
GeForce GTX 1080	7,550	29.98
Radeon VII	7,440	594.50
Quadro GP100	7,404	NA
GeForce RTX 2080 (Mobile)	7,403	NA

https://www.cpubenchmark.net/high_end_cpus.html



High End Hard Drive Chart

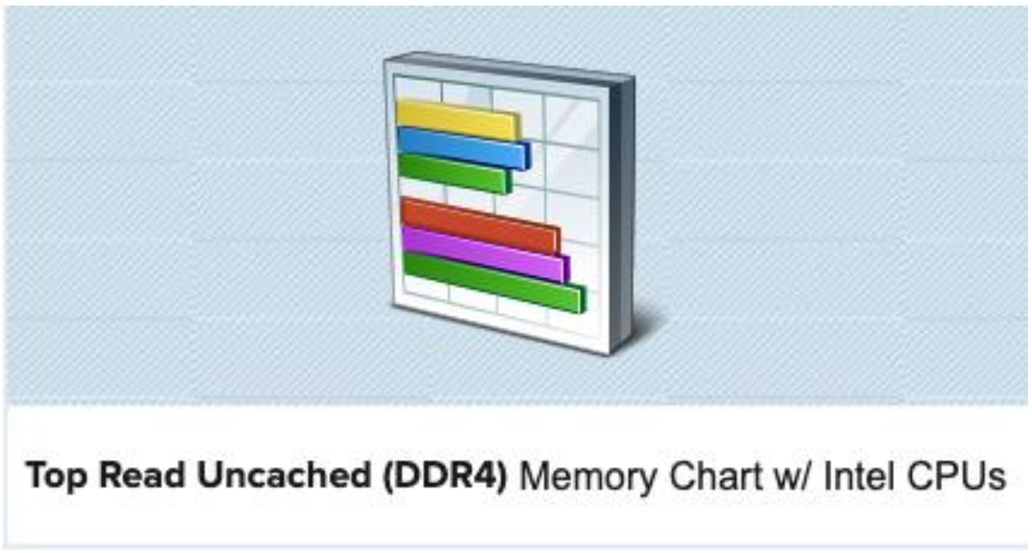
This chart is made using thousands of PerformanceTest benchmark results and is updated daily. These overall scores are calculated from three different tests measuring the read speed, write speed and seek time of hard disk drives. The chart contains drives from many of the major manufacturers such as Seagate, Western Digital (WDC), Hitachi, Maxtor and Samsung. The higher scoring drives in this chart are typically those with greater RPM values (10,000/15,000 RPM) or that are Solid State Drives.

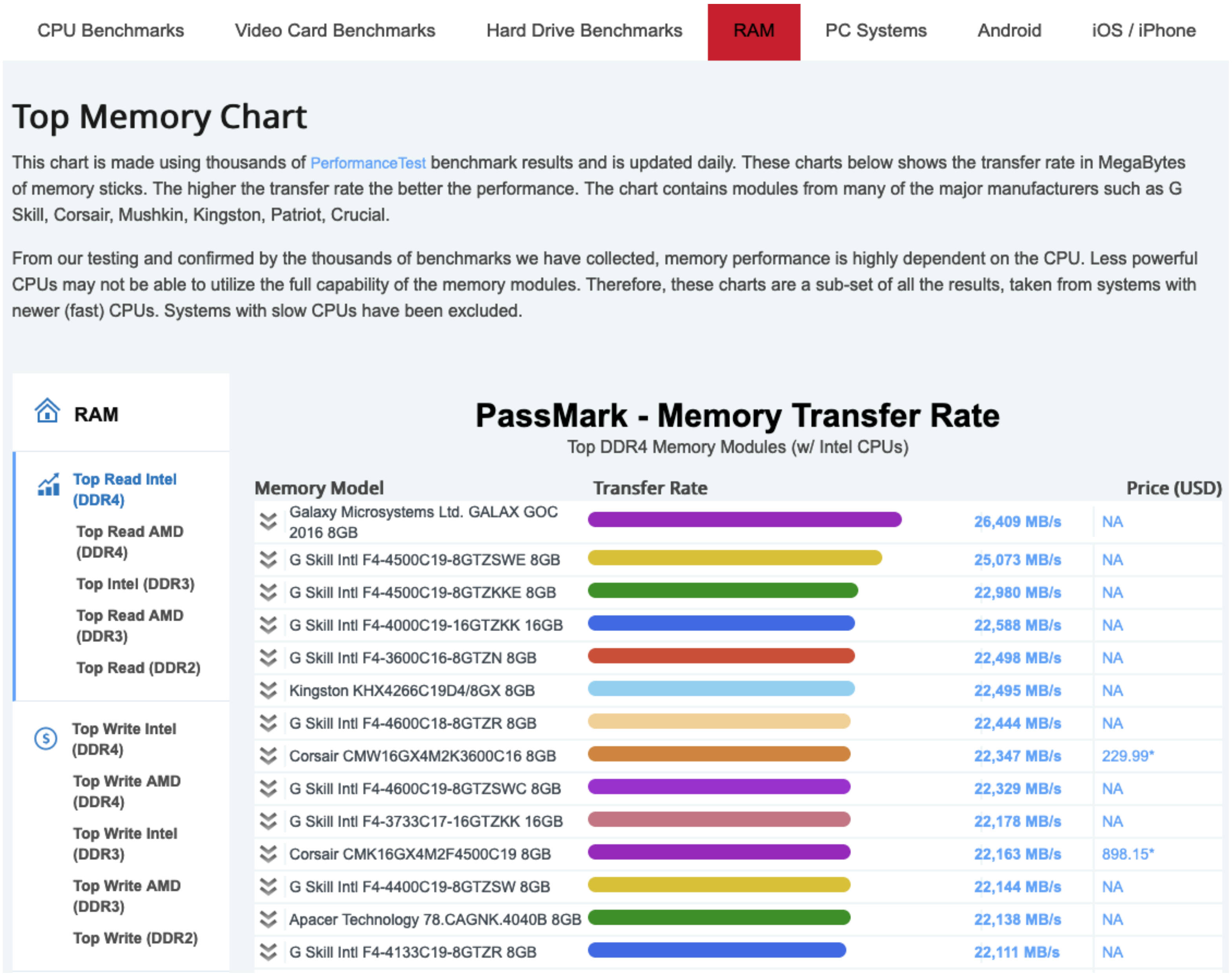
PassMark - Disk Rating

High End Drives

Drive	Disk Rating	Price (USD)
XPG GAMMIX S11 960GB	52,127	NA
CSSD-M2B1TPG3VNF	33,691	NA
NVMe Force MP600	33,382	NA
Seagate FireCuda 520 SSD ZP1000GM30002	33,154	NA
Sabrent Rocket 4.0 2TB	32,356	NA
Viper M.2 VP4100	31,744	219.99
Corsair Force MP600 1TB	31,191	259.99
Sabrent ROCKET 4.0 1TB	30,990	NA
Corsair Force MP600 2TB	30,581	399.99
Gigabyte AORUS NVMe Gen4 M.2 1TB	30,189	229.99
XPG GAMMIX S50	30,037	NA
ADATA SX8100NP	28,773	NA
Gigabyte AORUS NVMe Gen4 M.2 2TB	28,535	419.99
DIGISTOR 1TB	28,260	NA
RevuAhn NX2200A 1TB	27,800	NA
PM981a NVMe SED Samsung 512GB	27,759	NA

https://www.cpubenchmark.net/high_end_cpus.html


Top Read Uncached (DDR4) Memory Chart w/ Intel CPUs



Top Memory Chart

This chart is made using thousands of PerformanceTest benchmark results and is updated daily. These charts below shows the transfer rate in MegaBytes of memory sticks. The higher the transfer rate the better the performance. The chart contains modules from many of the major manufacturers such as G Skill, Corsair, Mushkin, Kingston, Patriot, Crucial.

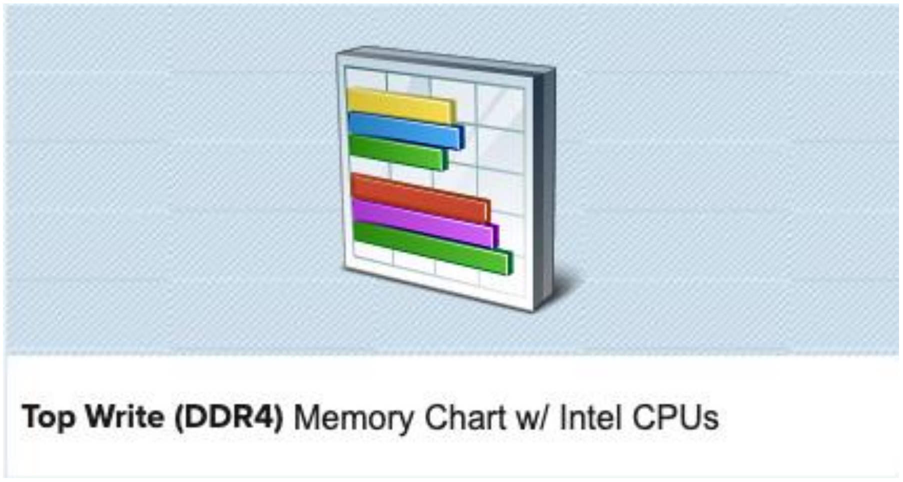
From our testing and confirmed by the thousands of benchmarks we have collected, memory performance is highly dependent on the CPU. Less powerful CPUs may not be able to utilize the full capability of the memory modules. Therefore, these charts are a sub-set of all the results, taken from systems with newer (fast) CPUs. Systems with slow CPUs have been excluded.

PassMark - Memory Transfer Rate

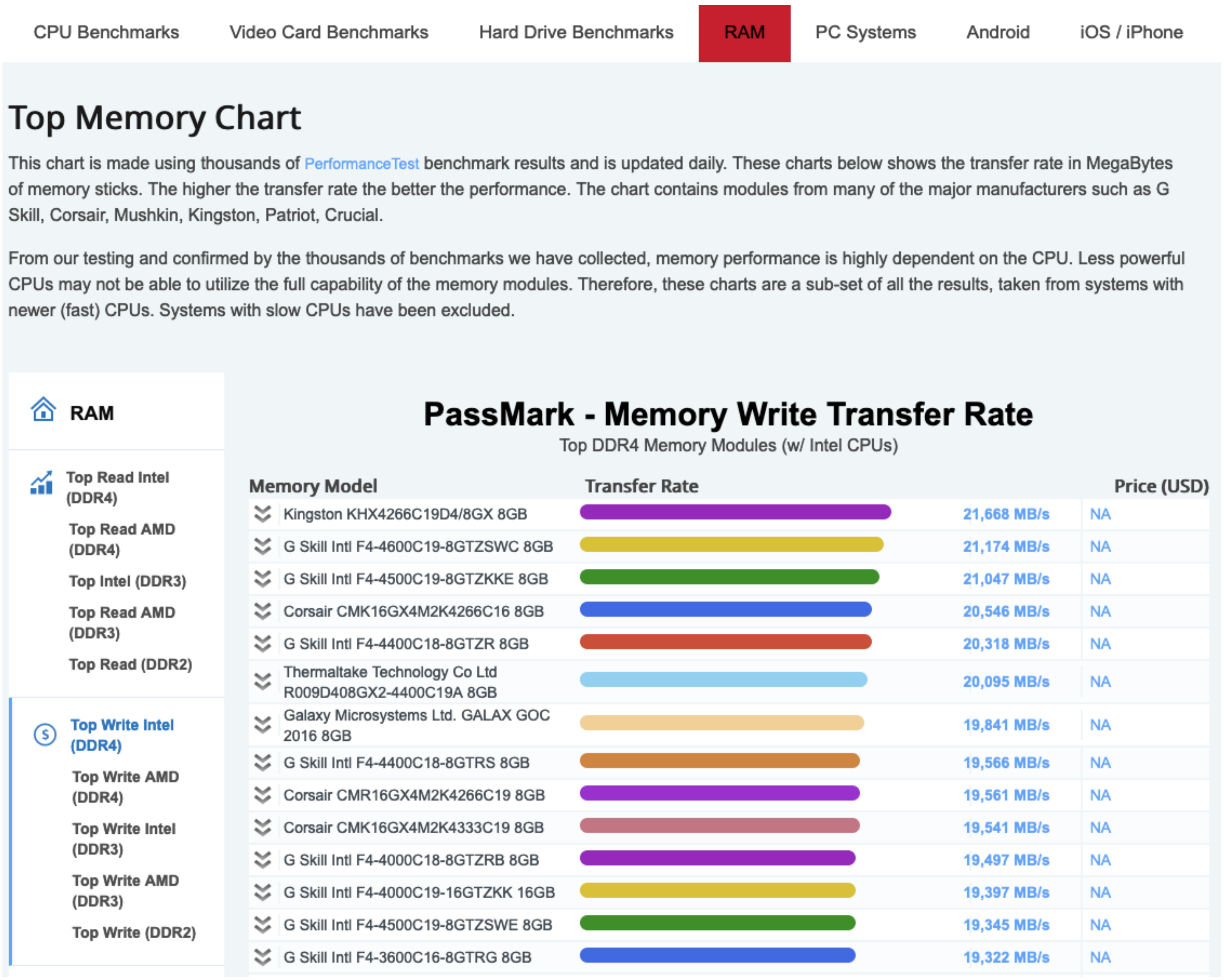
Top DDR4 Memory Modules (w/ Intel CPUs)

Memory Model	Transfer Rate	Price (USD)
Galaxy Microsystems Ltd. GALAX GOC 2016 8GB	26,409 MB/s	NA
G Skill Intl F4-4500C19-8GTZSWE 8GB	25,073 MB/s	NA
G Skill Intl F4-4500C19-8GTZKKE 8GB	22,980 MB/s	NA
G Skill Intl F4-4000C19-16GTZKK 16GB	22,588 MB/s	NA
G Skill Intl F4-3600C16-8GTZN 8GB	22,498 MB/s	NA
Kingston KHX4266C19D4/8GX 8GB	22,495 MB/s	NA
G Skill Intl F4-4600C18-8GTZR 8GB	22,444 MB/s	NA
Corsair CMW16GX4M2K3600C16 8GB	22,347 MB/s	229.99*
G Skill Intl F4-4600C19-8GTZSWC 8GB	22,329 MB/s	NA
G Skill Intl F4-3733C17-16GTZKK 16GB	22,178 MB/s	NA
Corsair CMK16GX4M2F4500C19 8GB	22,163 MB/s	898.15*
G Skill Intl F4-4400C19-8GTZSW 8GB	22,144 MB/s	NA
Apacer Technology 78.CAGNK.4040B 8GB	22,138 MB/s	NA
G Skill Intl F4-4133C19-8GTZR 8GB	22,111 MB/s	NA

https://www.cpubenchmark.net/high_end_cpus.html



Top Write (DDR4) Memory Chart w/ Intel CPUs



CPU Benchmarks | Video Card Benchmarks | Hard Drive Benchmarks | RAM | PC Systems | Android | iOS / iPhone

Top Memory Chart

This chart is made using thousands of PerformanceTest benchmark results and is updated daily. These charts below shows the transfer rate in MegaBytes of memory sticks. The higher the transfer rate the better the performance. The chart contains modules from many of the major manufacturers such as G Skill, Corsair, Mushkin, Kingston, Patriot, Crucial.

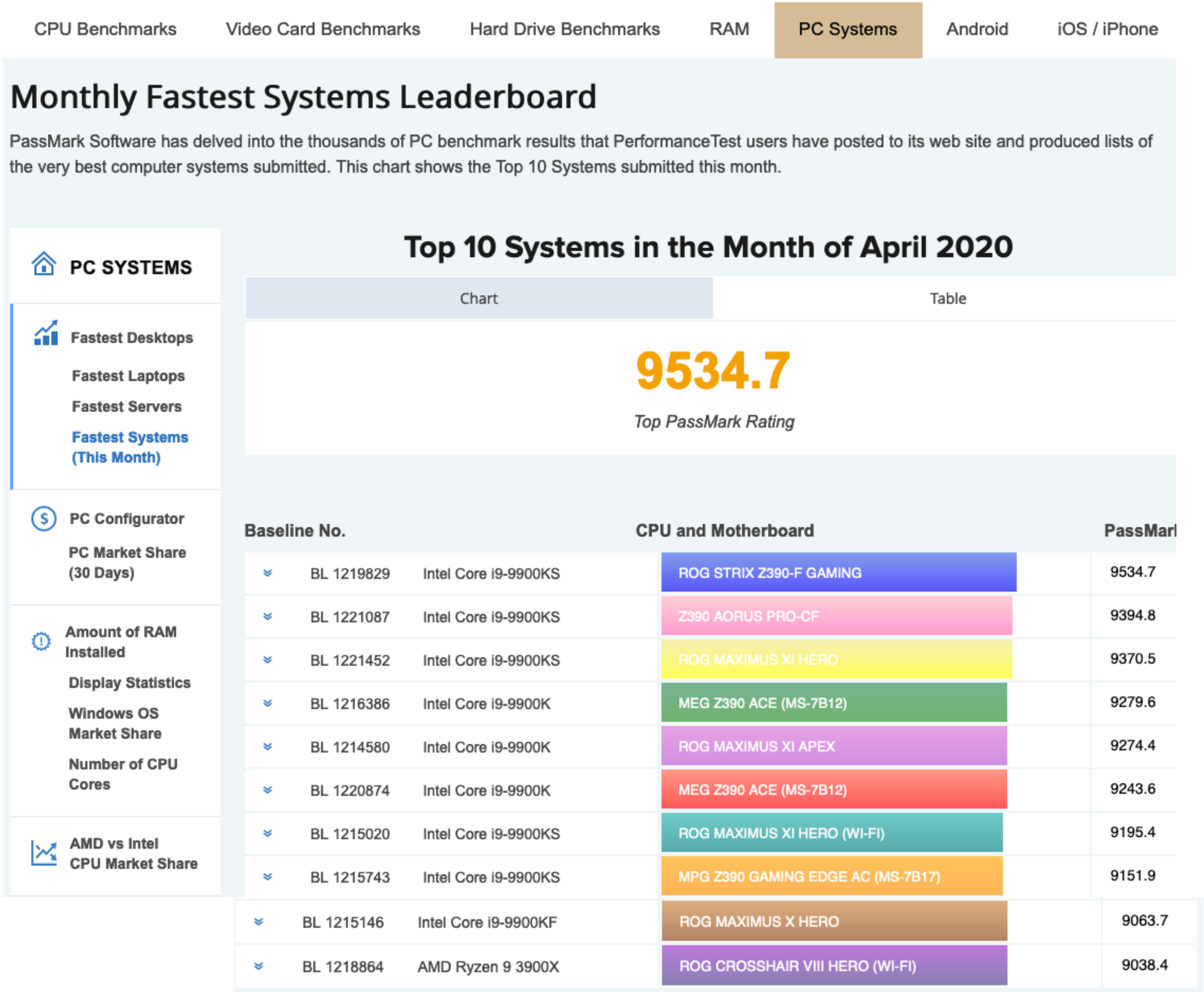
From our testing and confirmed by the thousands of benchmarks we have collected, memory performance is highly dependent on the CPU. Less powerful CPUs may not be able to utilize the full capability of the memory modules. Therefore, these charts are a sub-set of all the results, taken from systems with newer (fast) CPUs. Systems with slow CPUs have been excluded.

PassMark - Memory Write Transfer Rate

Top DDR4 Memory Modules (w/ Intel CPUs)

Memory Model	Transfer Rate		Price (USD)
Kingston KHX4266C19D4/8GX 8GB		21,668 MB/s	NA
G Skill Intl F4-4600C19-8GTZSWC 8GB		21,174 MB/s	NA
G Skill Intl F4-4500C19-8GTZKKE 8GB		21,047 MB/s	NA
Corsair CMK16GX4M2K4266C16 8GB		20,546 MB/s	NA
G Skill Intl F4-4400C18-8GTZR 8GB		20,318 MB/s	NA
Thermaltake Technology Co Ltd R009D408GX2-4400C19A 8GB		20,095 MB/s	NA
Galaxy Microsystems Ltd. GALAX GOC 2016 8GB		19,841 MB/s	NA
G Skill Intl F4-4400C18-8GTRS 8GB		19,566 MB/s	NA
Corsair CMR16GX4M2K4266C19 8GB		19,561 MB/s	NA
Corsair CMK16GX4M2K4333C19 8GB		19,541 MB/s	NA
G Skill Intl F4-4000C18-8GTZRB 8GB		19,497 MB/s	NA
G Skill Intl F4-4000C19-16GTZKK 16GB		19,397 MB/s	NA
G Skill Intl F4-4500C19-8GTZSWE 8GB		19,345 MB/s	NA
G Skill Intl F4-3600C16-8GTRG 8GB		19,322 MB/s	NA

https://www.cpubenchmark.net/high_end_cpus.html



Monthly Fastest Systems Leaderboard

PassMark Software has delved into the thousands of PC benchmark results that PerformanceTest users have posted to its web site and produced lists of the very best computer systems submitted. This chart shows the Top 10 Systems submitted this month.

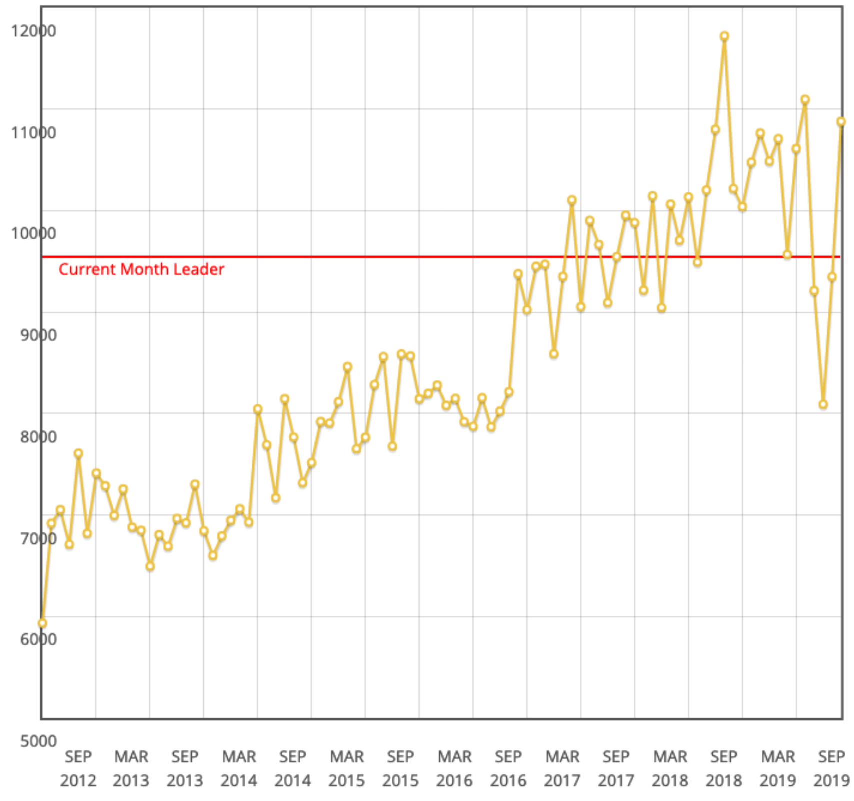
Top 10 Systems in the Month of April 2020

9534.7

Top PassMark Rating

	Baseline No.	CPU and Motherboard		PassMar
»	BL 1219829	Intel Core i9-9900KS	ROG STRIX Z390-F GAMING	9534.7
»	BL 1221087	Intel Core i9-9900KS	Z390 AORUS PRO-CF	9394.8
»	BL 1221452	Intel Core i9-9900KS	ROG MAXIMUS XI HERO	9370.5
»	BL 1216386	Intel Core i9-9900K	MEG Z390 ACE (MS-7B12)	9279.6
»	BL 1214580	Intel Core i9-9900K	ROG MAXIMUS XI APEX	9274.4
»	BL 1220874	Intel Core i9-9900K	MEG Z390 ACE (MS-7B12)	9243.6
»	BL 1215020	Intel Core i9-9900KS	ROG MAXIMUS XI HERO (WI-FI)	9195.4
»	BL 1215743	Intel Core i9-9900KS	MPG Z390 GAMING EDGE AC (MS-7B17)	9151.9
»	BL 1215146	Intel Core i9-9900KF	ROG MAXIMUS X HERO	9063.7
»	BL 1218864	AMD Ryzen 9 3900X	ROG CROSSHAIR VIII HERO (WI-FI)	9038.4

Past Monthly Leaders



https://www.cpubenchmark.net/high_end_cpus.html

https://study.com/academy/lesson/using-mips-flops-as-computer-performance-parameters.html

MIPs

MIPs (million instructions per second) is the general measurement or benchmark of how many instructions a processor can handle in a single second. Despite how useful this idea might seem, it is not commonly used anymore because there is no proper way of measuring MIPs. In general, a MIPs rating was only used as a basic rule of thumb for computer performance, since a higher number did not mean much for most real-world situations. In the business world, however, being able to calculate a MIPs rating allowed businesses to know the cost of computing from the servers they were using.

FLOPs

The clock speed of a processor is measured in megahertz and gigahertz, but that by itself is not an accurate way to gauge computer performance. **FLOPs (floating-point operations per second)** is yet another necessary factor needed to help measure the performance of a processor as shown in Figure 1. A **floating point number** is a number that has floating decimal points, such as 0.008. A FLOPs benchmark only measures the floating point operations and not the integers, which means it too cannot solely gauge computer performance either.

It makes more sense to measure today's processor performance in FLOPs because clock speed frequencies do not truly measure raw performance. A FLOPs measurement most accurately represents computer performance since floating point math is standard in a variety of programs or processes that we use today. It is especially useful for scientific and real-time applications, such as gaming or image processing.

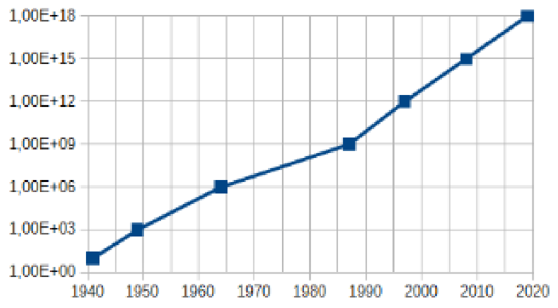


Figure 1: Computer Performance Evolution Over Time (FLOPs)

https://towardsdatascience.com/20-popular-machine-learning-metrics-part-1-classification-regression-evaluation-metrics-1ca3e282a2ce

M

towards data science

DATA SCIENCE · MACHINE LEARNING · PROGRAMMING · VISUALIZATION · AI · VIDEO · ABOUT | CONTRIBUTE

20 Popular Machine Learning Metrics. Part 1: Classification & Regression Evaluation Metrics

An introduction to the most important metrics for evaluating classification, regression, ranking, vision, NLP, and deep learning models.

Shervin Minaee Follow
Oct 28, 2019 · 11 min read ★

Note: This post has two parts. **In the first part** (current post), I will talk about 10 metrics that are widely used for evaluating classification and regression models. And **in the second part** I will talk about 10 metrics which are used to evaluate ranking, computer vision, NLP, and deep learning models.