Levels of Computing

Joseph T. Wunderlich, Ph.D.

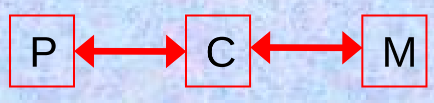
Levels of Computing

1	Embedded
2	PC
3	PC Server or Workstation
4	Mini computer
5	Super Computer

Computer Architectures

- P = Processor (CPU)
- C = Cache
- M = Main Memory (RAM)

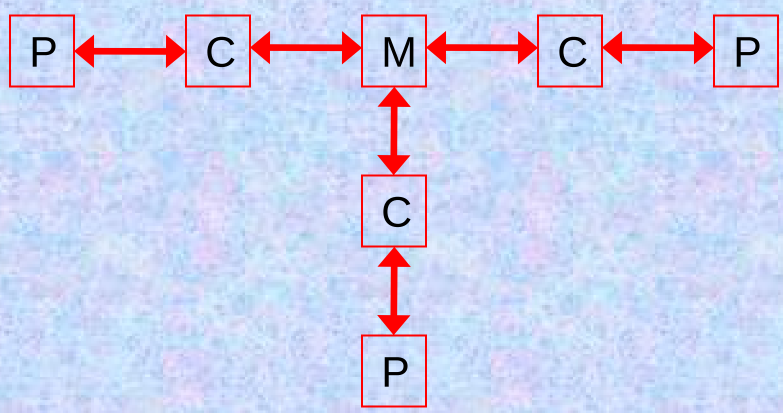
Simple Single Processor



Computer Architectures

P = Processor (CPU)
C = Cache
M = Main Memory (RAM)

- SMP (**S**ymmetric **M**ulti-**P**rocessing)

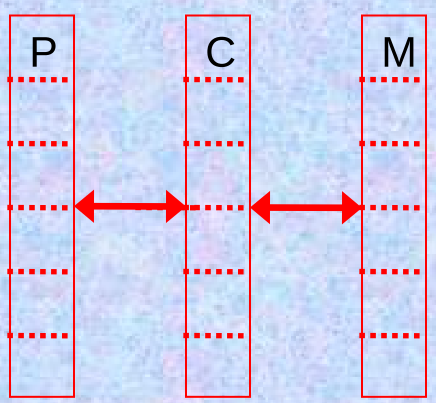


Computer Architectures

P = Processor (CPU)
C = Cache
M = Main Memory (RAM)

Vector Register

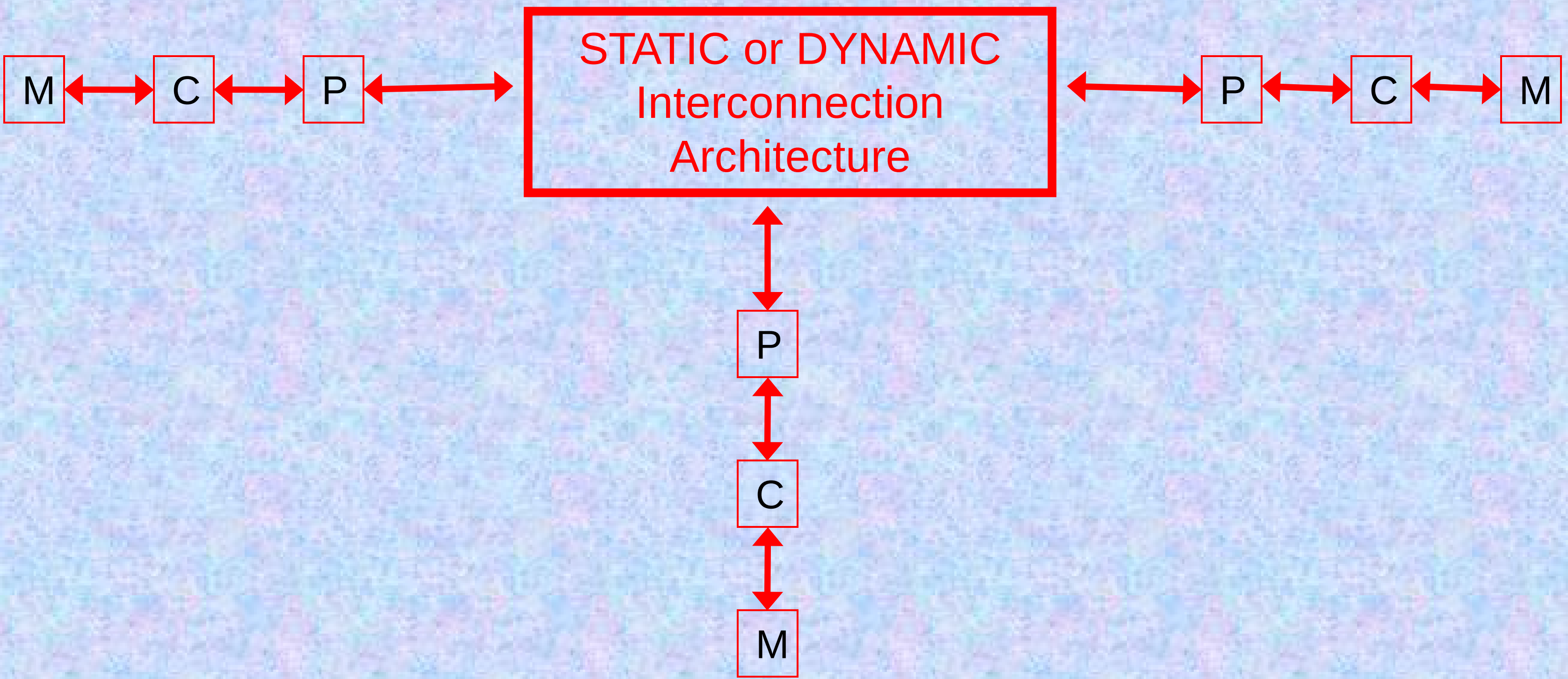
- Multiple functional units in Processor for arithmetic and logic
- Multiple data elements in Cache and main Memory



Computer Architectures

P = Processor (CPU)
C = Cache
M = Main Memory (RAM)

MPP (**M**assively **P**arallel **P**rocessing)



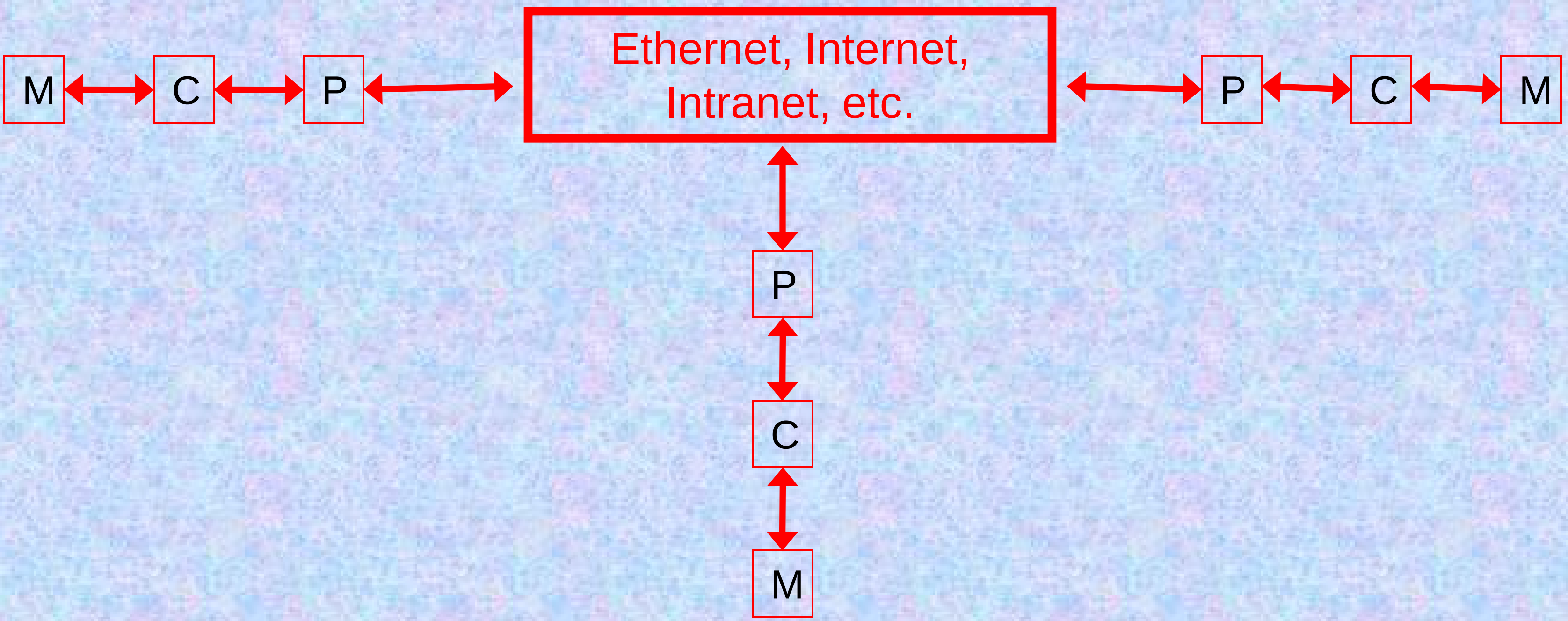
Computer Architectures

P = Processor (CPU)

C = Cache

M = Main Memory (RAM)

Large network dedicated to a single task



LEVEL	ARCHITECTURE
Embedded	SMP, MPP, or Vector-Register
PC	SMP
PC Server or Workstation	SMP
Mini computer	SMP
Super Computer	SMP, MPP, Vector-Register, *or* Large network dedicated to a single task

LEVEL	APPLICATIONS
Embedded	Real-Time control: Automobiles, Appliances, factory automation, Aerospace
PC	General-purpose “low-end” computing
PC Server or Workstation	LAN server for ~100 people, 3-D simulations, VLSI circuit design
Mini computer	LAN server for ~500 people
Super Computer	SMP: LAN, WAN, or Internet server for 1000’s of people, Air traffic control, NYSE Vector-Register: Matrix-intensive Grand Challenge App’s MPP: Grand Challenge App’s, Chess Large network: Human Genome

LEVEL	CHARACTERISTICS
Embedded	Cheap, small, and can be extremely fast – but typically not. May be hardened for industry or space
PC	Faster than typical embedded, but otherwise relatively slow. O(~$3000)
PC Server or Workstation	Faster O(~$3000 to ~$15,000)
Mini computer	Very fast O(~$100,000)
Super Computer	Extremely fast O(~$1,000,000 to ~$10,000,000)

LEVEL	EXAMPLE DEVICES
Embedded	• Microcontroller (Intel, Motorola, PIC’s) • Microprocessor (Intel, Motorola, PowerPC) • Application Specific IC’s (ASIC’s) • Programmable Logic Controllers (PLC’s)
PC	Microprocessor (Intel, Motorola, PowerPC)
PC Server or Workstation	Multiple microprocessors (Intel, Motorola, PowerPC, Sparc) … Silicon Graphics Terminals, SUN or IBM RS6000 workstations
Mini computer	IBM AS400, Amdahl, HP, Hitachi
Super Computer	SMP: IBM S/390 Vector-Register: CRAY MPP: IBM SP2 Large network: PC’s everywhere

LEVEL	OPERATING SYSTEMS
Embedded	None or custom **Possibly a real-time OS**
PC	Windows, DOS, CP/M, OS2, MAC OS, B, Linux, etc.
PC Server or Workstation	Windows NT, UNIX, AIX
Mini computer	UNIX, MVS, VMS, OS 390
Super Computer	SMP: UNIX, MVS, VMS, OS 390 Vector-Register: custom vector OS MPP: custom distributed OS Large network: PC OS's

MICROPROCESSOR	MICROCONTROLLER
For general-purpose computing	Intentionally simple for single-chip embedded applications
Can be **C**omplex **I**nstruction **S**et **C**omputing (CISC)	Intentionally Reduced Instruction Set Computing (RISC)
Both integer and Floating-Point calculations	Intentionally simple integer-only calculations
Large Address Spaces (Plus virtual Addressing)	Can put all data and instructions in on-chip RAM
Versatility	On-chip device control capabilities: DAC, ADC, PWM

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

Microprocessors (with Floating-Point) vs. Microcontrollers (only Integers)

INTEGER NUMBER RANGES

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 = **-32,768to 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

FLOATING POINT NUMBER RANGES

IEEE single precision (32-bit) BFP **-1*10^38 to 1*10^38**
IEEE double precision (64-bit) BFP **–1*10^308 to 1*10^308**

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

Microprocessors (with Floating-Point) vs. Microcontrollers (only Integers)

INTEGER NUMBER PRECISION (i.e., smallest number)

8-bit unsigned: **1**
8-bit signed: **1**

16-bit unsigned: **1**
16-bit signed: **1**

32-bit unsigned: **1**
32-bit signed: **1**

n-bit unsigned: **1**
n-bit signed: **1**

FLOATING POINT NUMBER RANGES

IEEE single precision (32-bit) BFP: **10^-39**
IEEE double precision (64-bit) BFP: **10^-308**

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

Consider a “Minimal Computer Architecture” ……

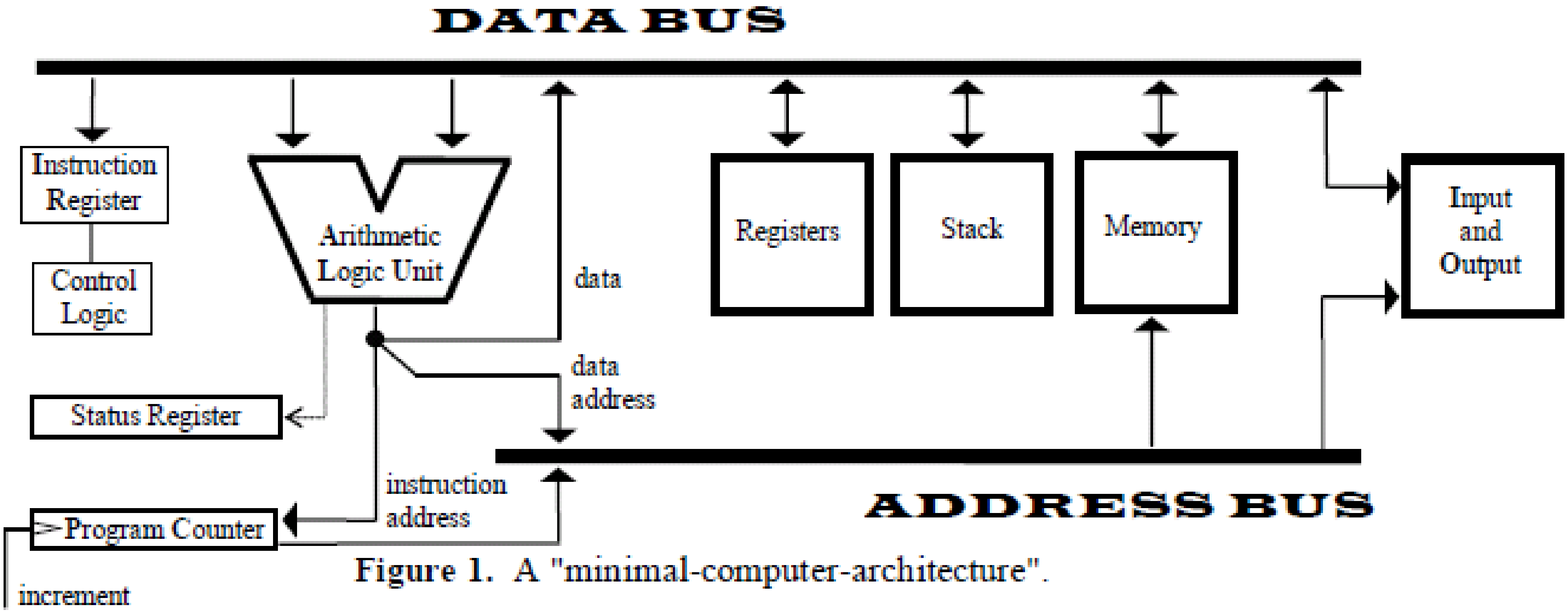
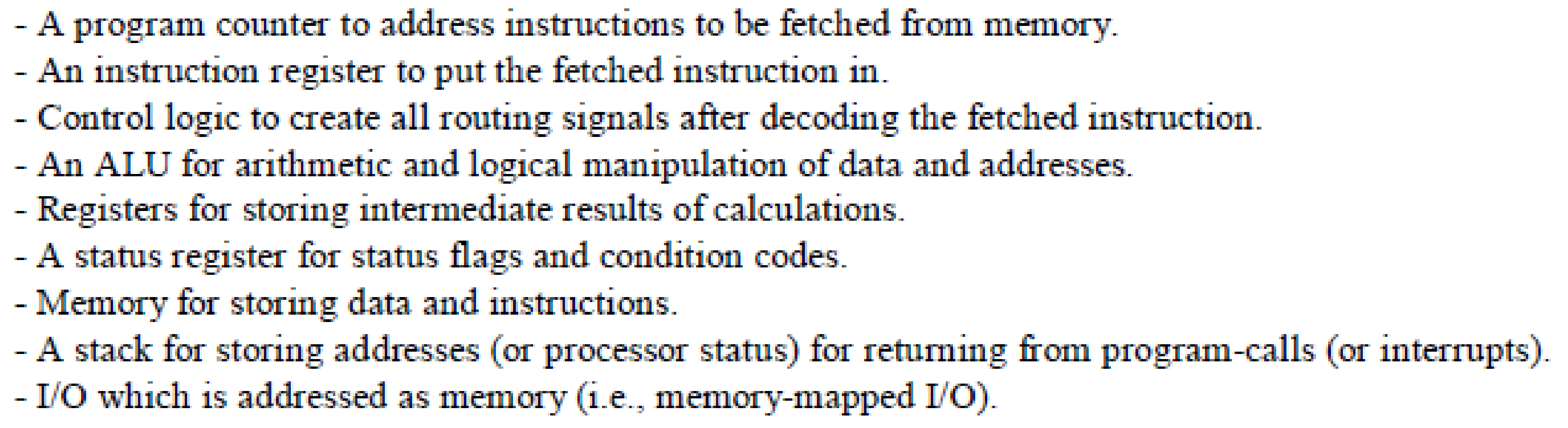


Figure 1. A "minimal-computer-architecture".



- 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).

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

Microprocessors vs. Microcontrollers

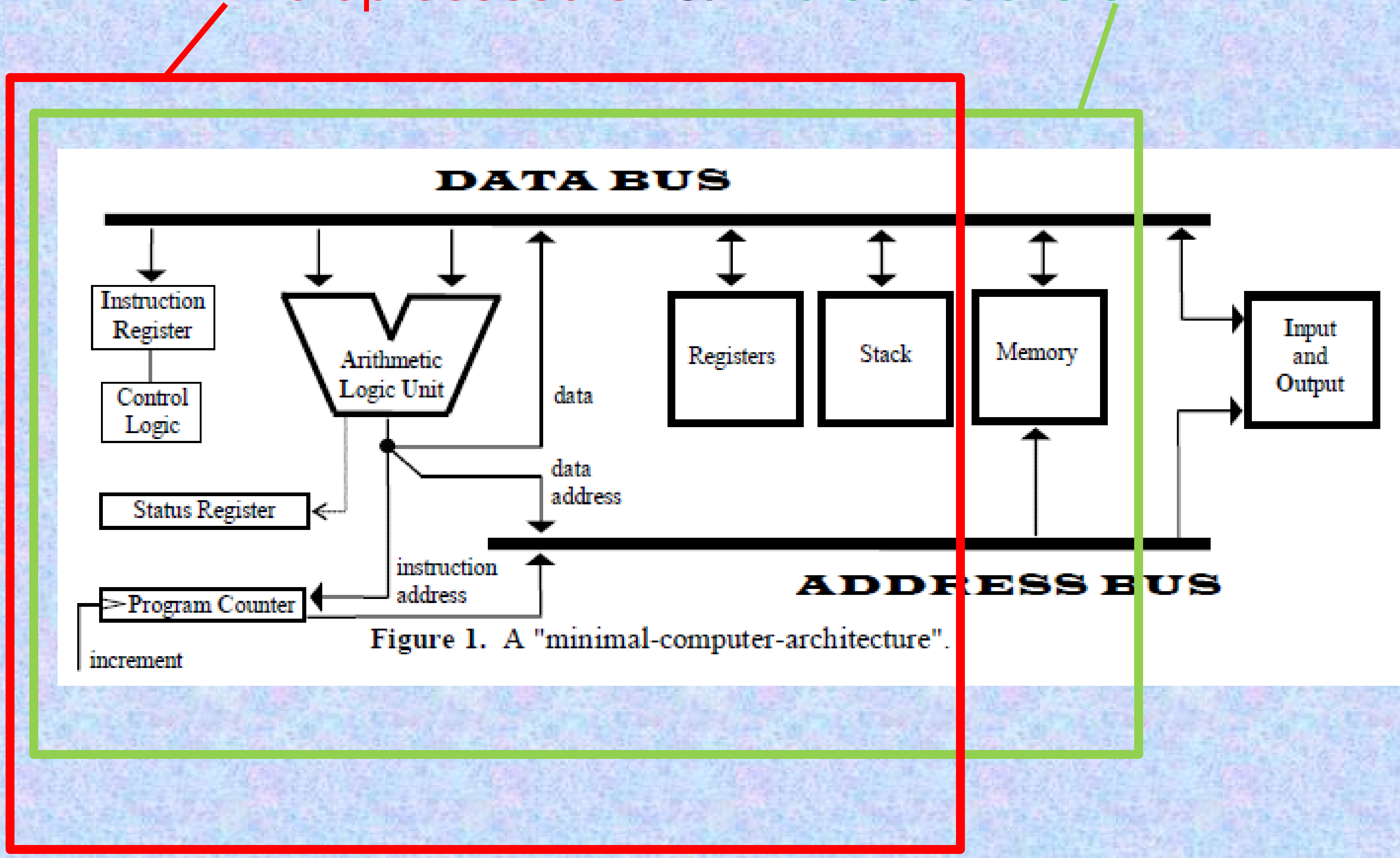
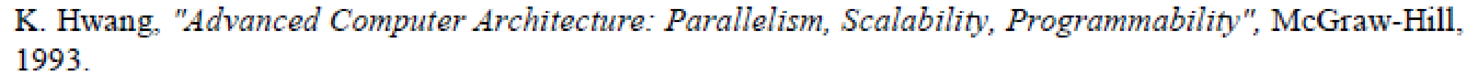


Figure 1. A "minimal-computer-architecture".

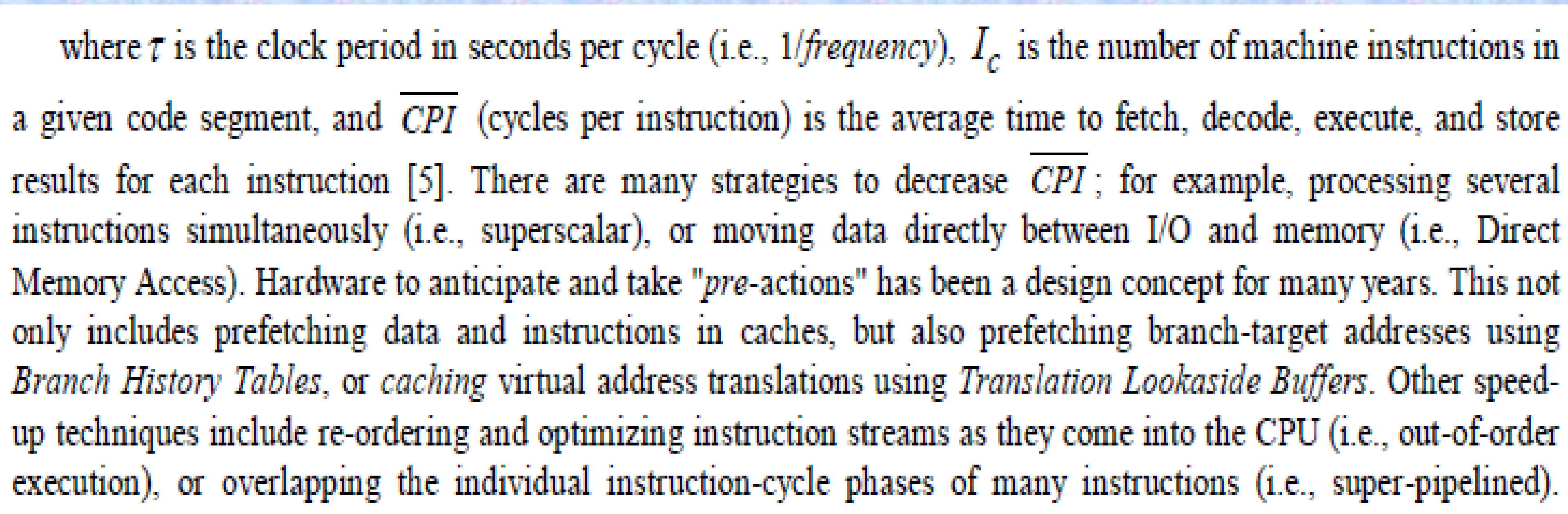
Wunderlich, J.T. (1999). **Focusing on the blurry distinction between microprocessors and microcontrollers.** In *Proceedings of 1999 ASEE Annual Conference & Exposition, Charlotte, NC*: (session 3547), [CD-ROM]. ASEE Publications.

Consider the time it takes to execute a segment of code

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



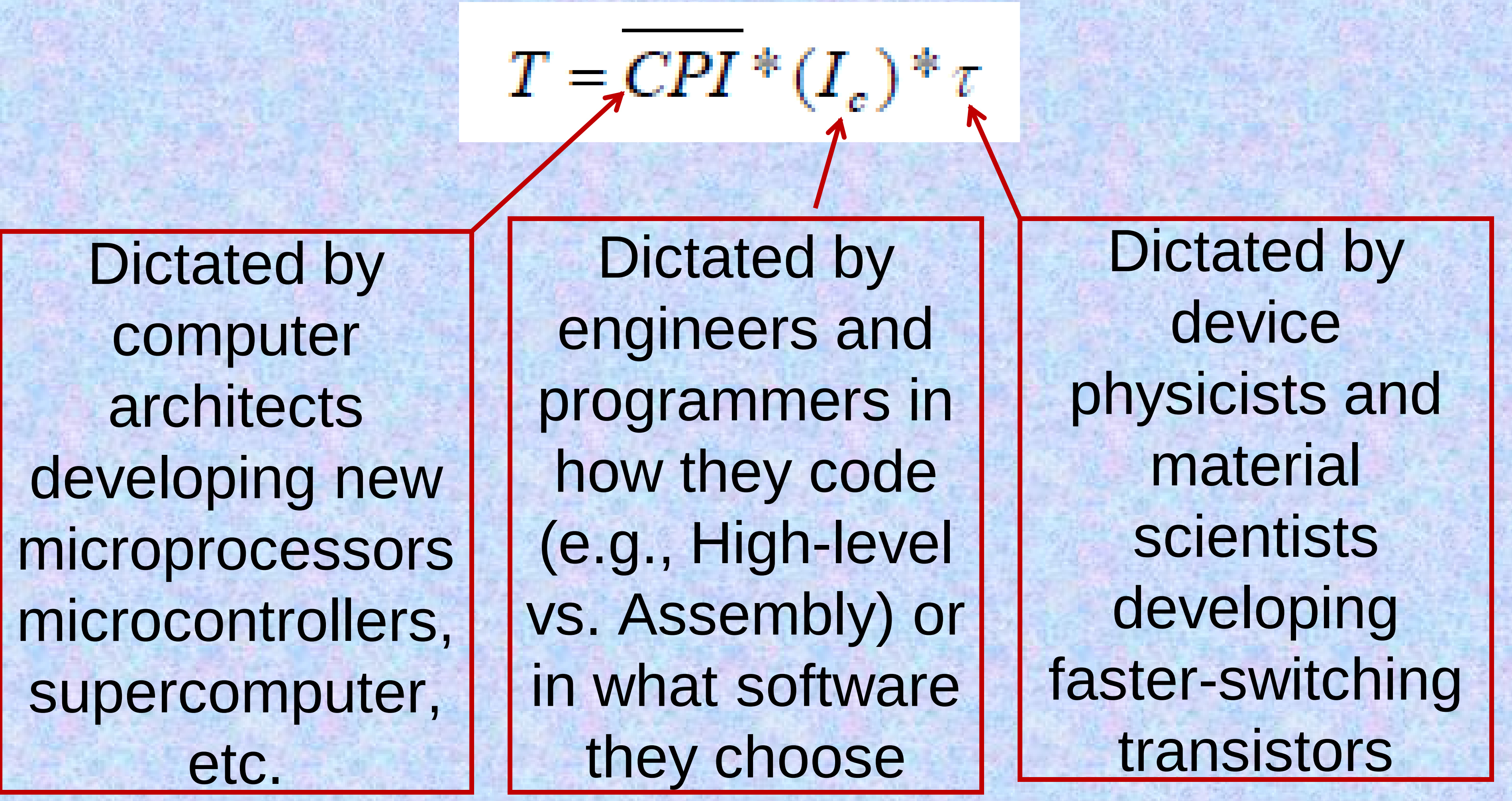
K. Hwang, *"Advanced Computer Architecture: Parallelism, Scalability, Programmability"*, McGraw-Hill, 1993.



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-pipelined).

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

Consider the time it takes to execute a segment of code



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

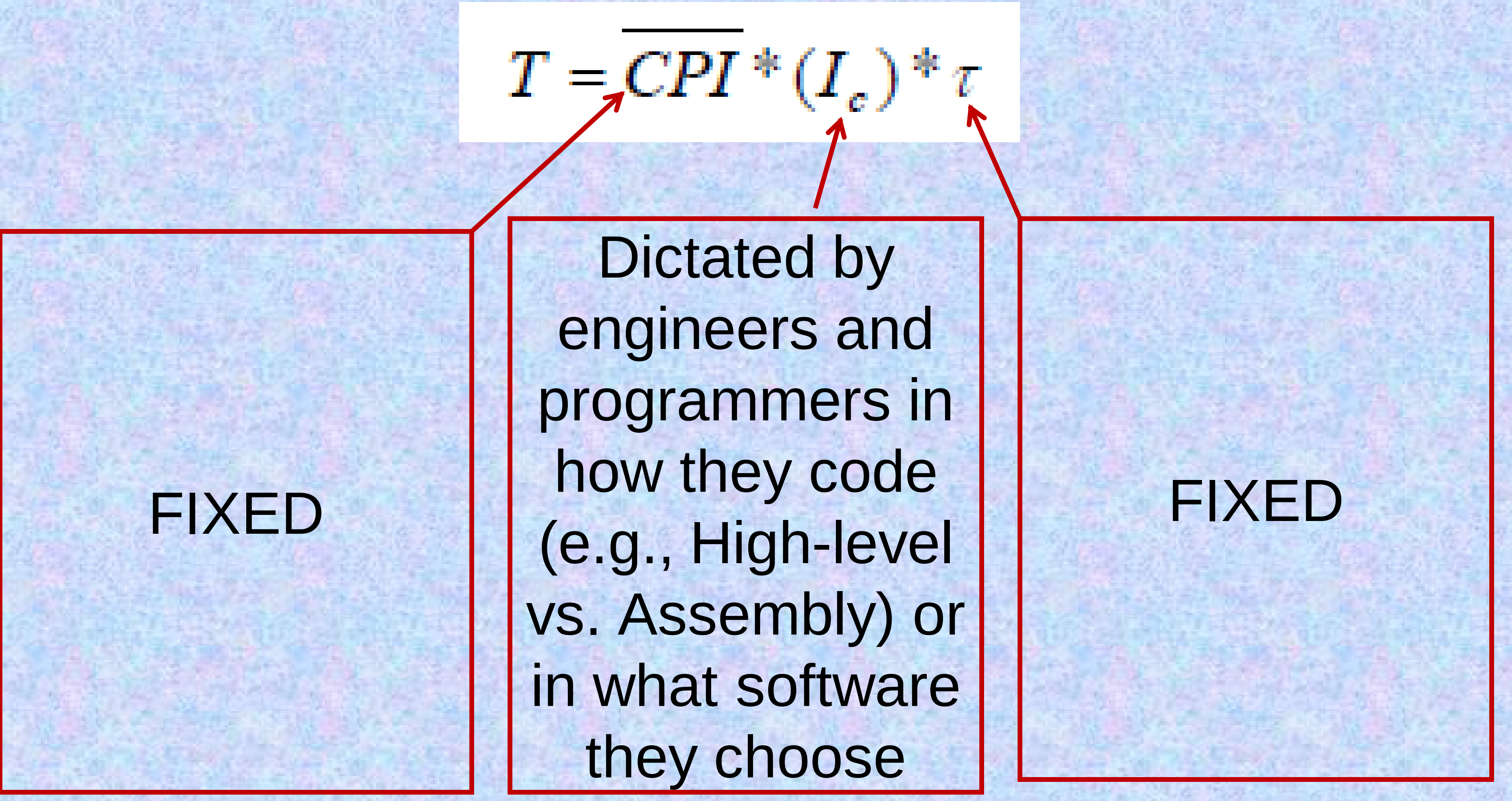
Dictated by computer architects developing new microprocessors microcontrollers, supercomputer, etc.

Dictated by engineers and programmers in how they code (e.g., High-level vs. Assembly) or in what software they choose

Dictated by device physicists and material scientists developing faster-switching transistors

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

For most applications ………..



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

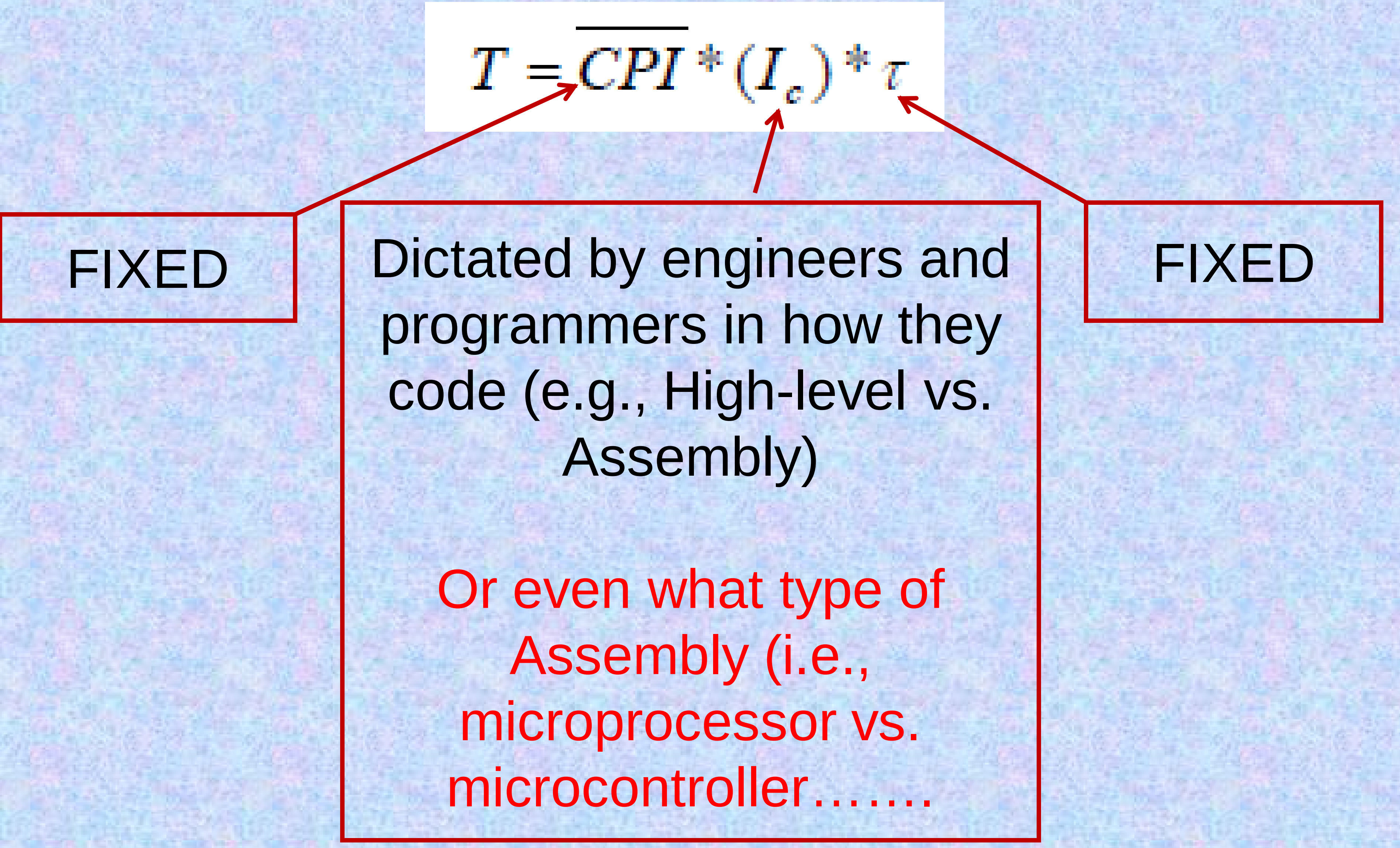
FIXED

Dictated by engineers and programmers in how they code (e.g., High-level vs. Assembly) or in what software they choose

FIXED

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

For most applications ……….



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

FIXED

Dictated by engineers and programmers in how they code (e.g., High-level vs. Assembly)

Or even what type of Assembly (i.e., microprocessor vs. microcontroller…….

FIXED

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

Microprocessors vs. Microcontrollers

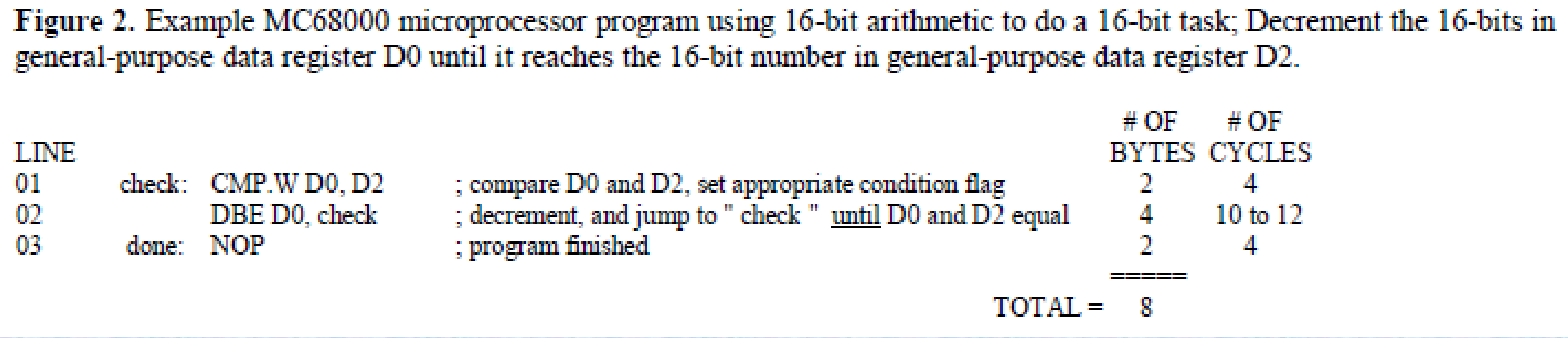


Figure 2. Example MC68000 microprocessor program using 16-bit arithmetic to do a 16-bit task; Decrement the 16-bits in general-purpose data register D0 until it reaches the 16-bit number in general-purpose data register D2.

```
                                                                               # 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
```

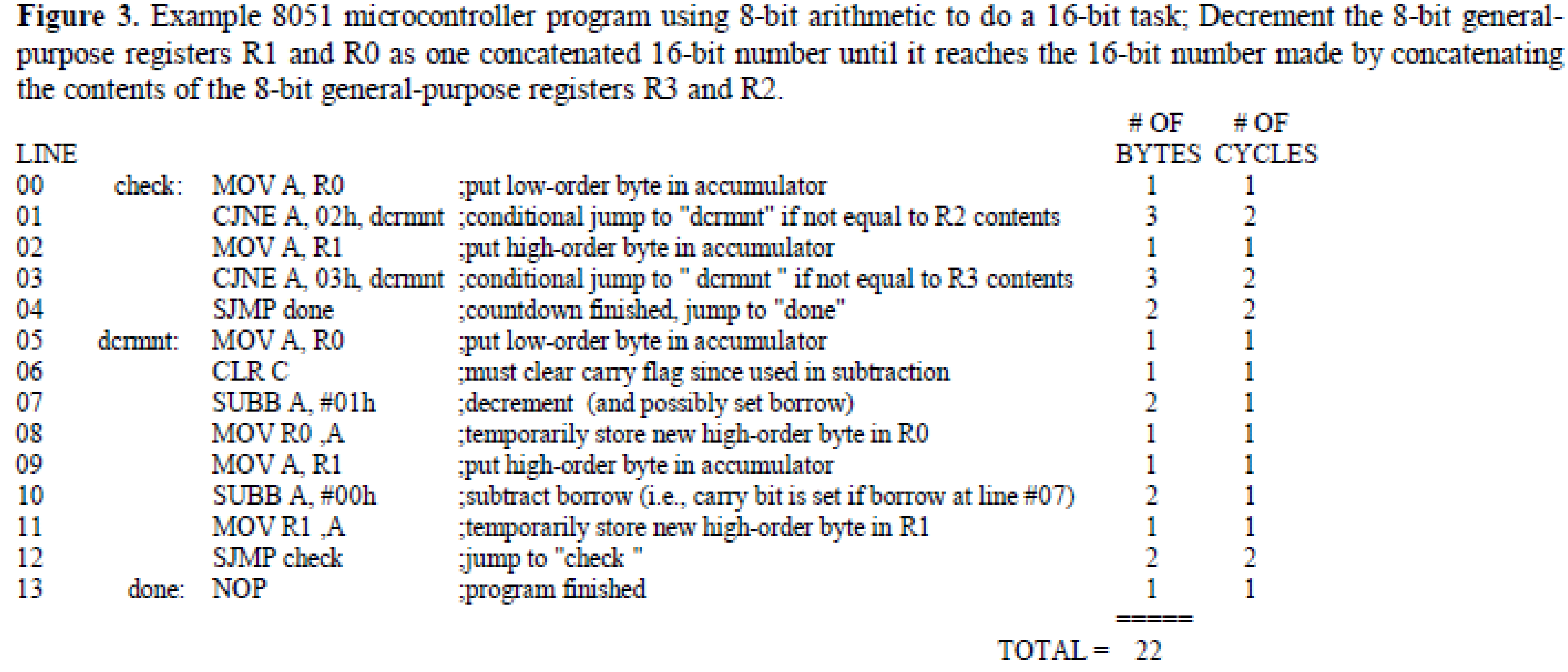
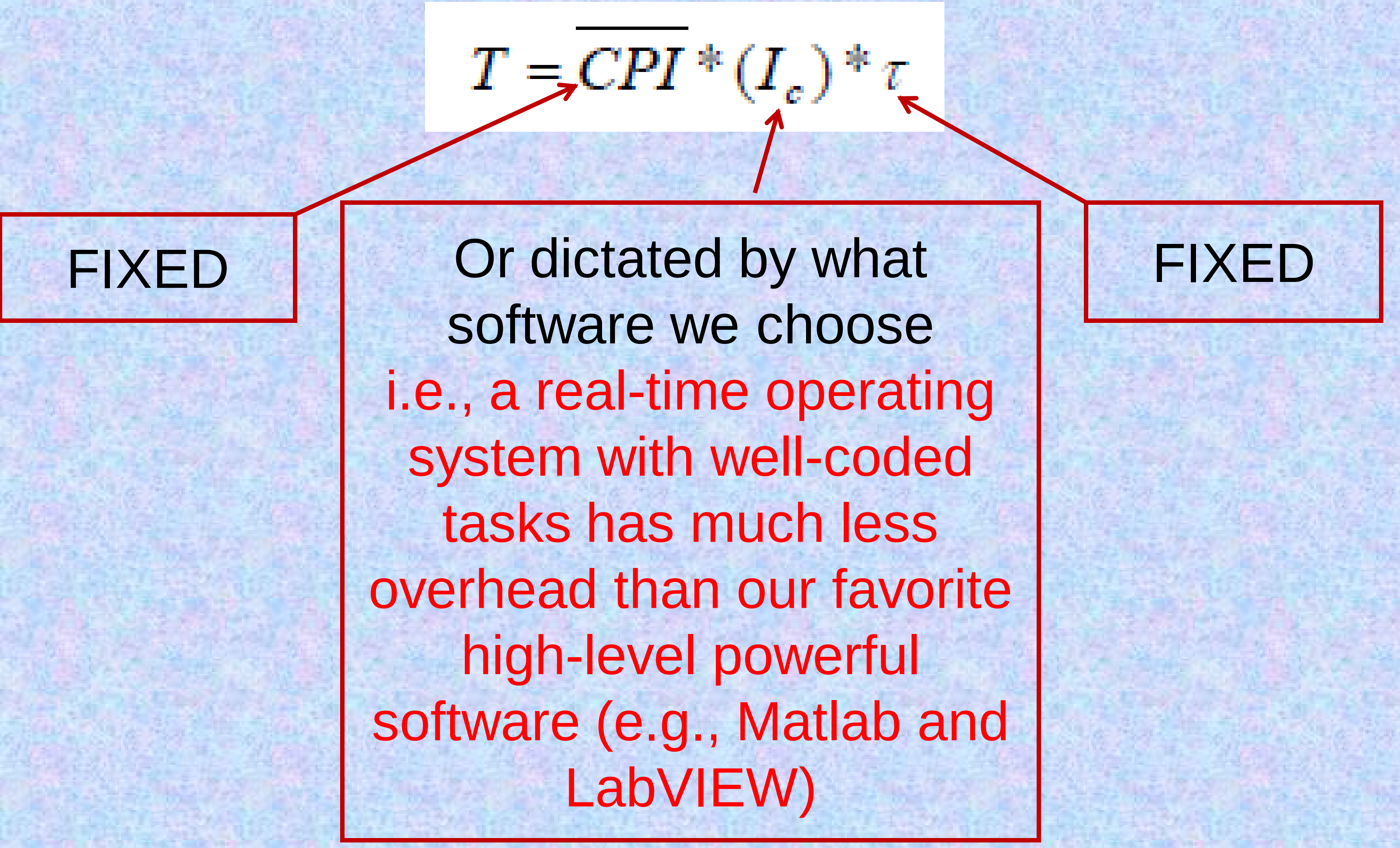


Figure 3. Example 8051 microcontroller program using 8-bit arithmetic to do a 16-bit task; Decrement the 8-bit general-purpose registers R1 and R0 as one concatenated 16-bit number until it reaches the 16-bit number made by concatenating the contents of the 8-bit general-purpose registers R3 and R2.

```
                                                                                # OF    # OF
LINE                                                                           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
```

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

For most applications ………..



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

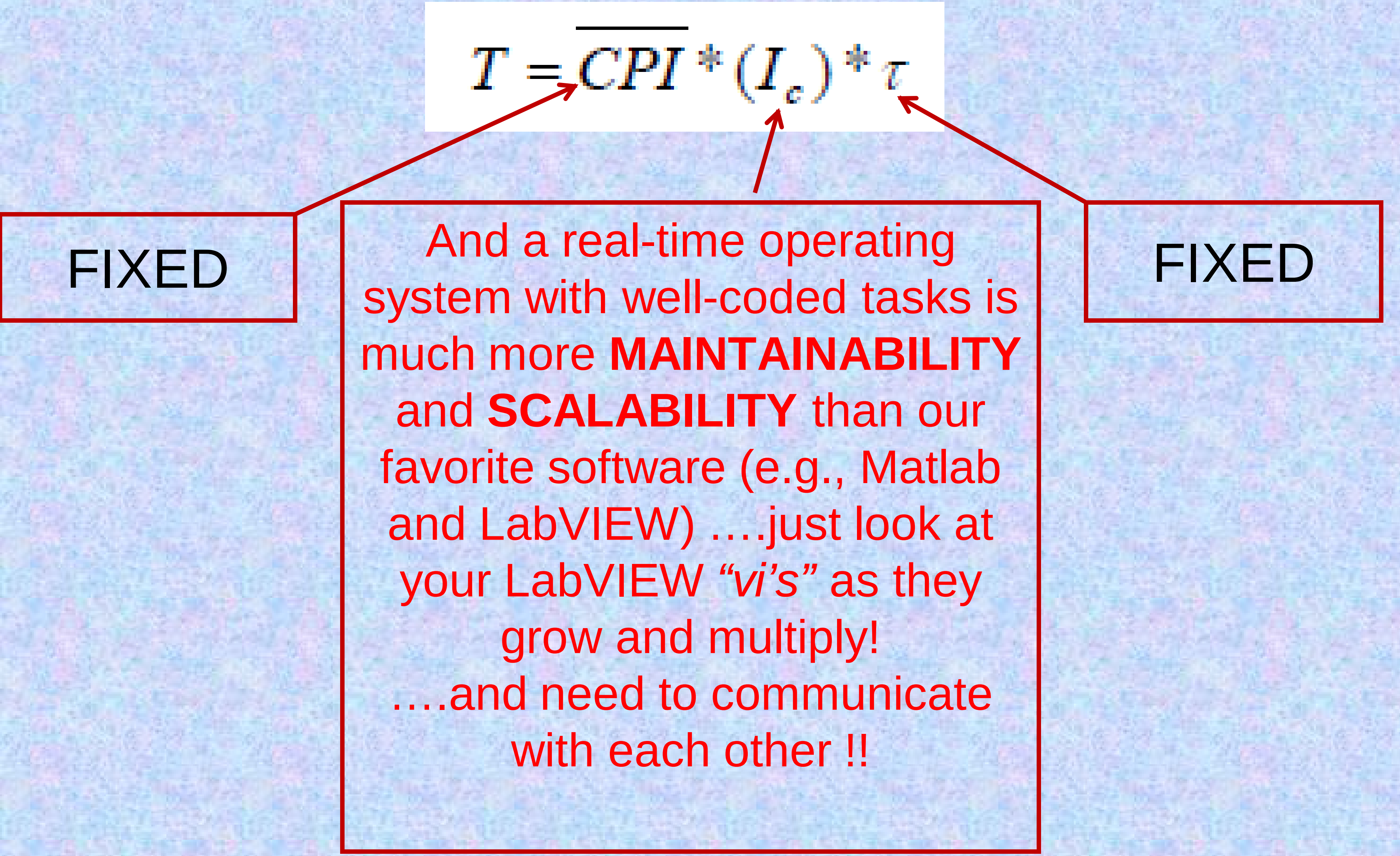
FIXED

Or dictated by what software we choose i.e., a real-time operating system with well-coded tasks has much less overhead than our favorite high-level powerful software (e.g., Matlab and LabVIEW)

FIXED

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

For most applications ………..



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

FIXED

And a real-time operating system with well-coded tasks is much more **MAINTAINABILITY** and **SCALABILITY** than our favorite software (e.g., Matlab and LabVIEW) ….just look at your LabVIEW *“vi’s”* as they grow and multiply!
….and need to communicate with each other !!

FIXED

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

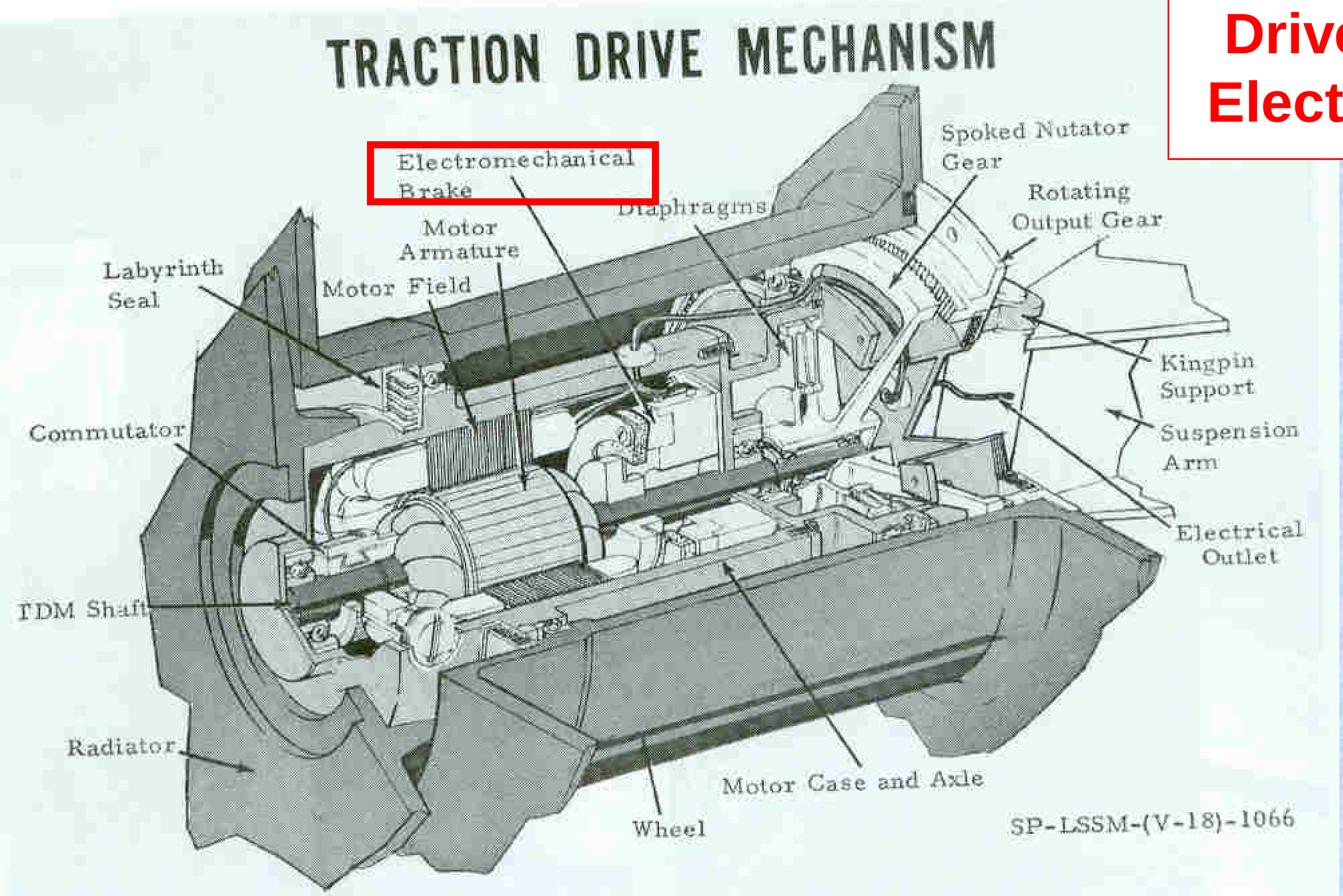
Microprocessors vs. Microcontrollers

READ MORE AT:

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

"Lunar Roving Vehicle" (LRV)

Drive Controller Electronics (DCE)



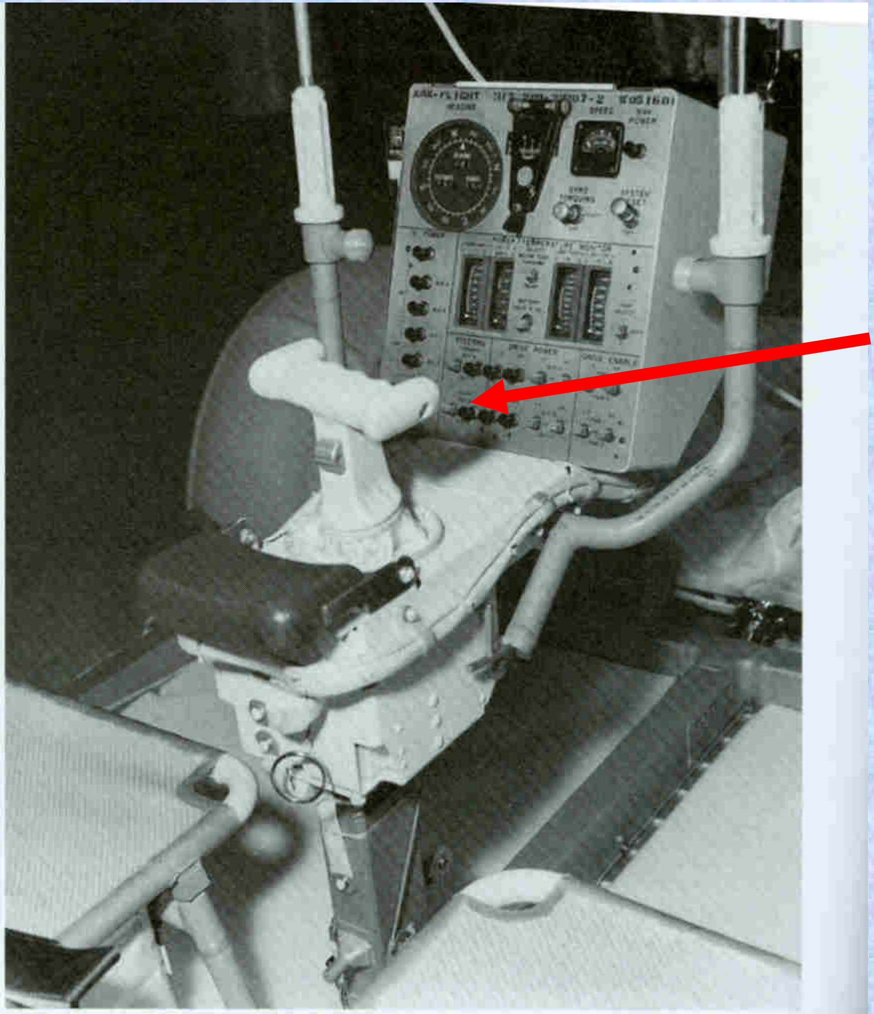
Each of the four traction drive motors had a rated output of 0.25 horsepower, with a combined output of one horsepower for the LRV. The drives were completely sealed to prevent damage from lunar dust. (NASA/MSFC)

Motor Control

and Braking

Image from: Young, A.H. ***Lunar and planetary rovers: the wheels of Apollo and the quest for mars***, Springer; 1 edition, August 1, 2006.

"Lunar Roving Vehicle" (LRV)



The Control and Display Console of the Qualification Unit was clearly marked "Non-Flight." The inboard hand-holds with light colored grips were vital for properly seating on the Lunar Rover in 1/6 gravity in the astronaut's pressure suits. The left hand-hold also served as a mount for the Low-Gain Antenna, and the right hand-hold served as the mount for the 16mm Data Acquisition Camera (DAC). The Sun Shadow Device is in the stowed position to the right of the Heading Indicator. (NASA)

Drive Controller Electronics (DCE)

Steering ("T-Handle")

- Pivot forward = *accelerate forward*
- Pivot rearward = *accelerate backward*
- Pivot left = *turn left*
- Pivot right = *turn right*
- Slide handle backward = *apply the brake and disengage the throttle*
- Slide controller all the way back = *engage the parking brake*
- *Switch on handle activated reverse*

Image from: Young, A.H. ***Lunar and planetary rovers: the wheels of Apollo and the quest for mars***, Springer; 1 edition, August 1, 2006.
Steering info from: http://science.howstuffworks.com/lunar-rover.htm/printable

“Lunar Roving Vehicle” (LRV)

Drive Controller Electronics (DCE)

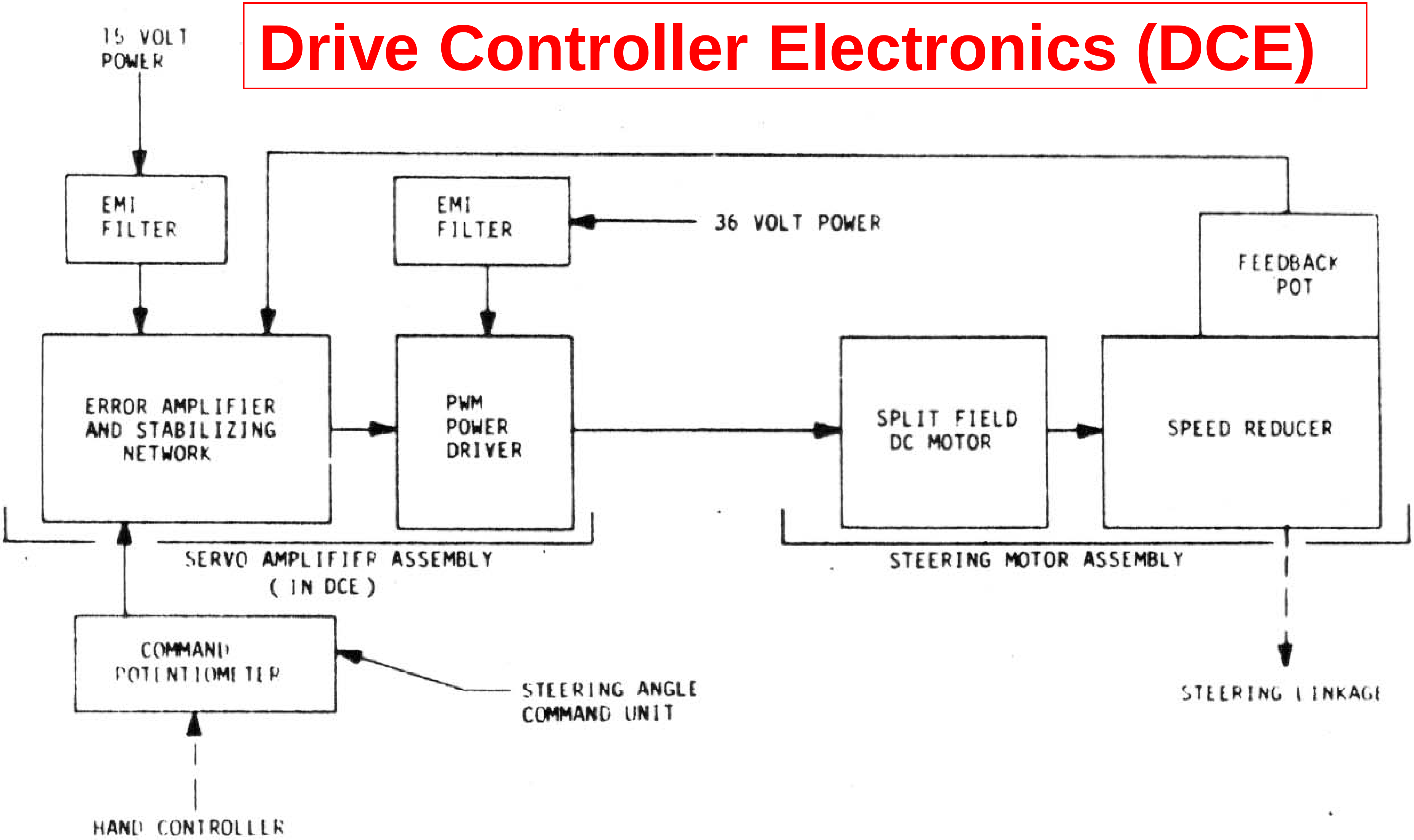
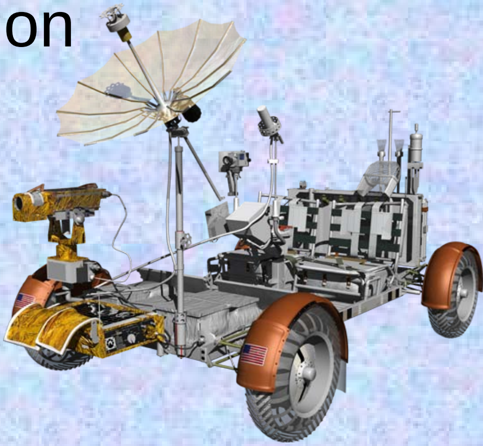


Image from: http://history.nasa.gov/alsj/LRV_OpsNAS8-25145Pt1.pdf

“Lunar Roving Vehicle” (LRV) *Computers*

NASA functional specifications for LRV Navigation:

1. Able to navigate to a predetermined location
2. Output speed and distance traveled
3. Calculate a <u>shortest path</u> back to Lander



LRV Navigation subsystem components:

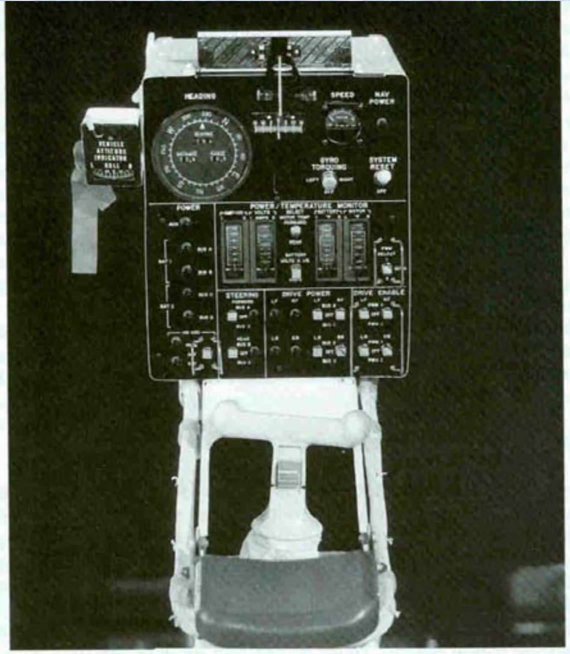
1. Directional Gyroscope Unit (DGU)*Lear Seigler Model 9010*
2. Sun angle measurement
3. Integrated Position Indicator (IPI) *by Abrams Instrument co.*
4. Four odometers (one for each independent wheel drive)
5. Custom Signal Processing Unit (SPU) by *Boeing co.*

   - This was the computer.

Source: Young, A.H. ***Lunar and planetary rovers: the wheels of Apollo and the quest for mars,*** Springer; 1 edition, August 1, 2006.

“Lunar Roving Vehicle” (LRV)

User Interface



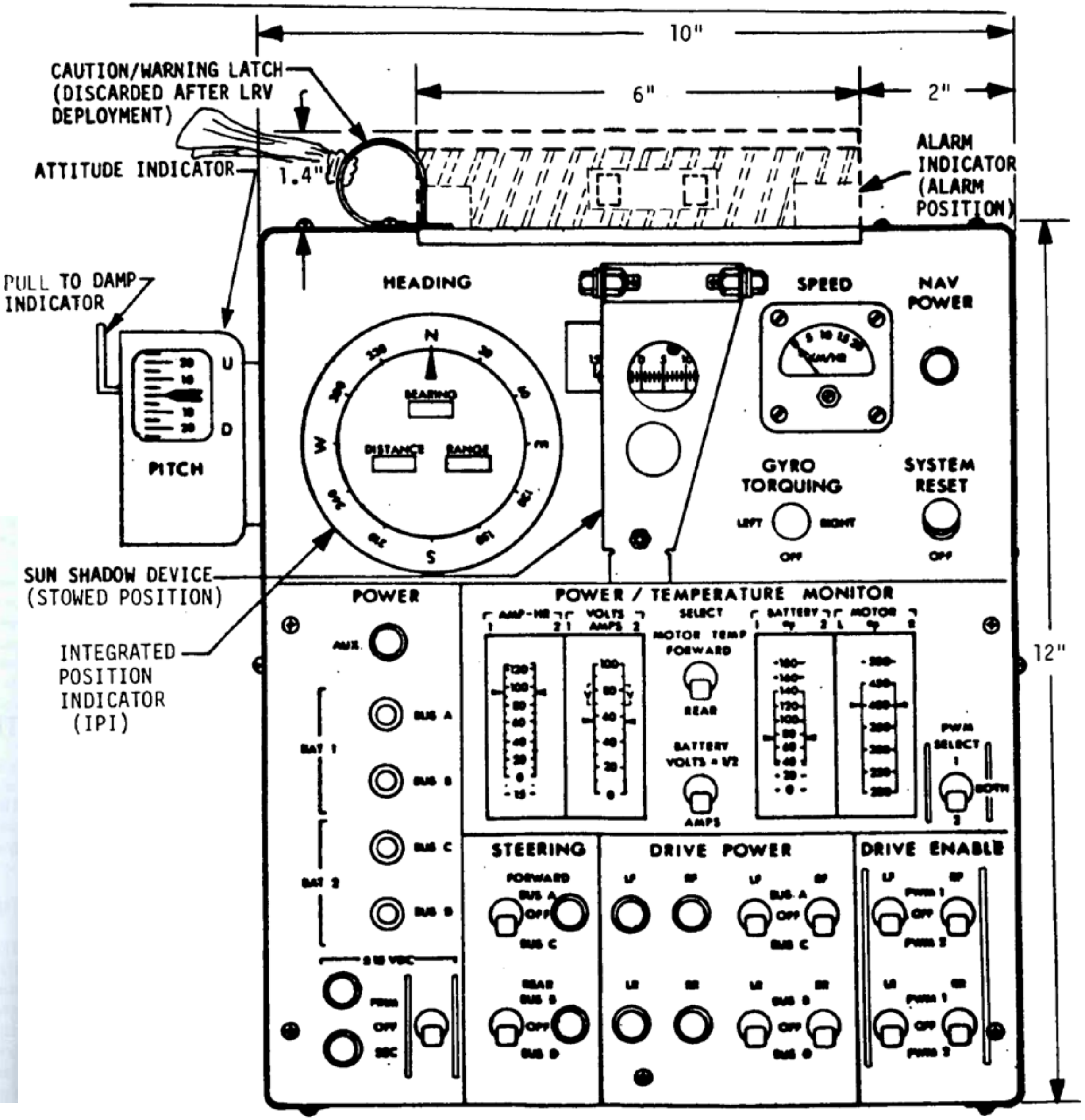


Image from: http://www.lpi.usra.edu/lunar/documents/NTRS/collection2/NASA_TM_X_66816.pdf

Mars Rovers *Computers*

"Sojourner" had a 0.1-MHz Intel 80C85 CPU with 512 Kbytes of RAM and 176 Kbytes of flash memory. (embedded system)

The **MER** vehicles ***"Spirit and "Opportunity"*** have a 20-MHz RAD6000 CPU with 128 Mbytes of RAM and 256 Mbytes of flash memory. (embedded system)

And the **MSL** ***"Mars Science Lab"*** vehicle will have a 200-MHz RAD750 PowerPC with 256 Mbytes of RAM and 512 Mbytes of flash memory. (embedded system)

The MER and MSL vehicles use the **VxWorks REAL-TIME operating system** and run many parallel tasks continuously

See more at:

Bajracharya, M., Maimone, M.W., and Helmick, D. (2008). ***Autonomy for Mars Rovers: Past, Present, and Future.*** **In Computer: December, 2008. (pp. 44-50). IEEE Press.**

VxWorks tutorial: http://www.cross-comp.com/instr/pages/embedded/VxWorksTutorial.aspx

Wunderbot IV

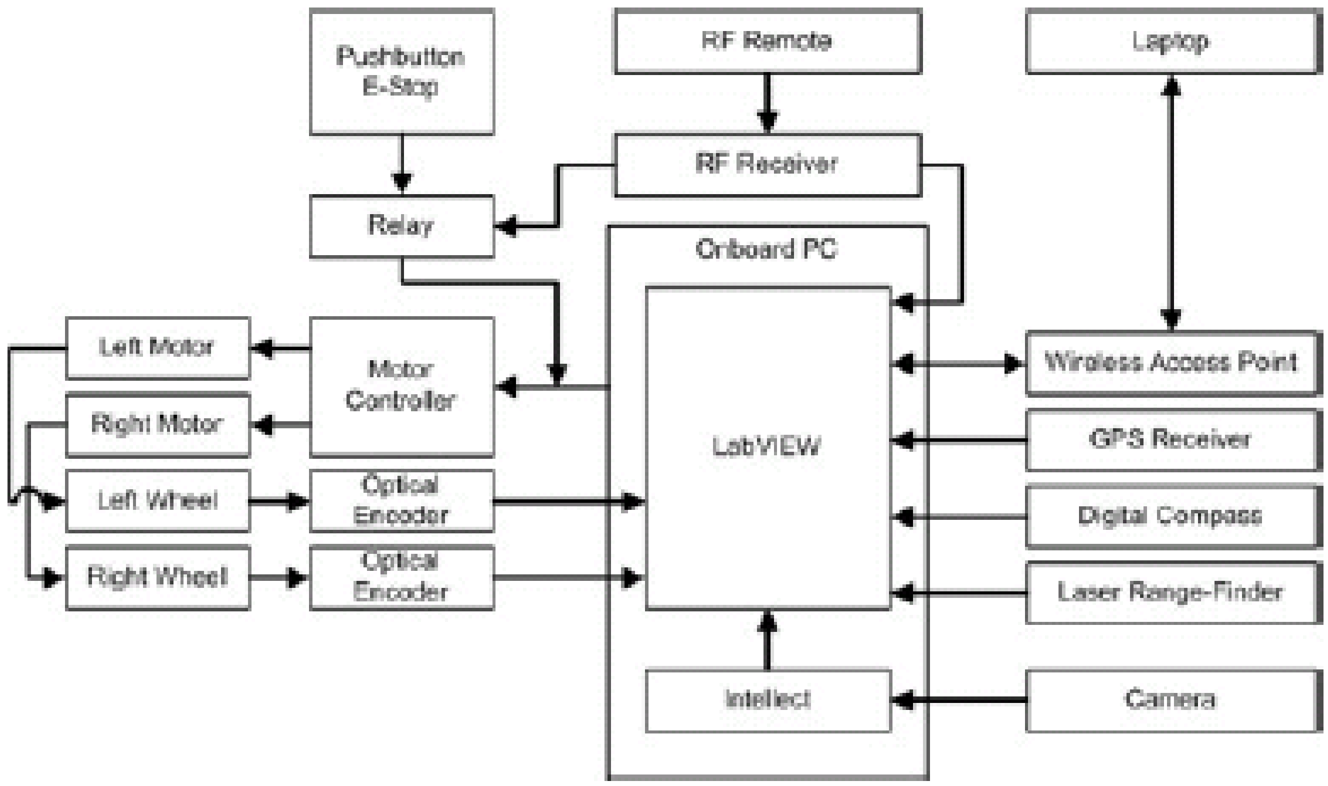


Fig. 8. Block diagram of Wunderbot IV subsystems.


Wunderbot 4

SOURCE: Painter, J. and Wunderlich, J.T. (2008). **Wunderbot IV: autonomous robot for international competition**. In *Proceedings of the 12th World Multi-Conference on Systemics, Cybernetics and Informatics: WMSCI 2008, Orlando, FL*: (pp. 62-67). And HERE

Wunderbot IV

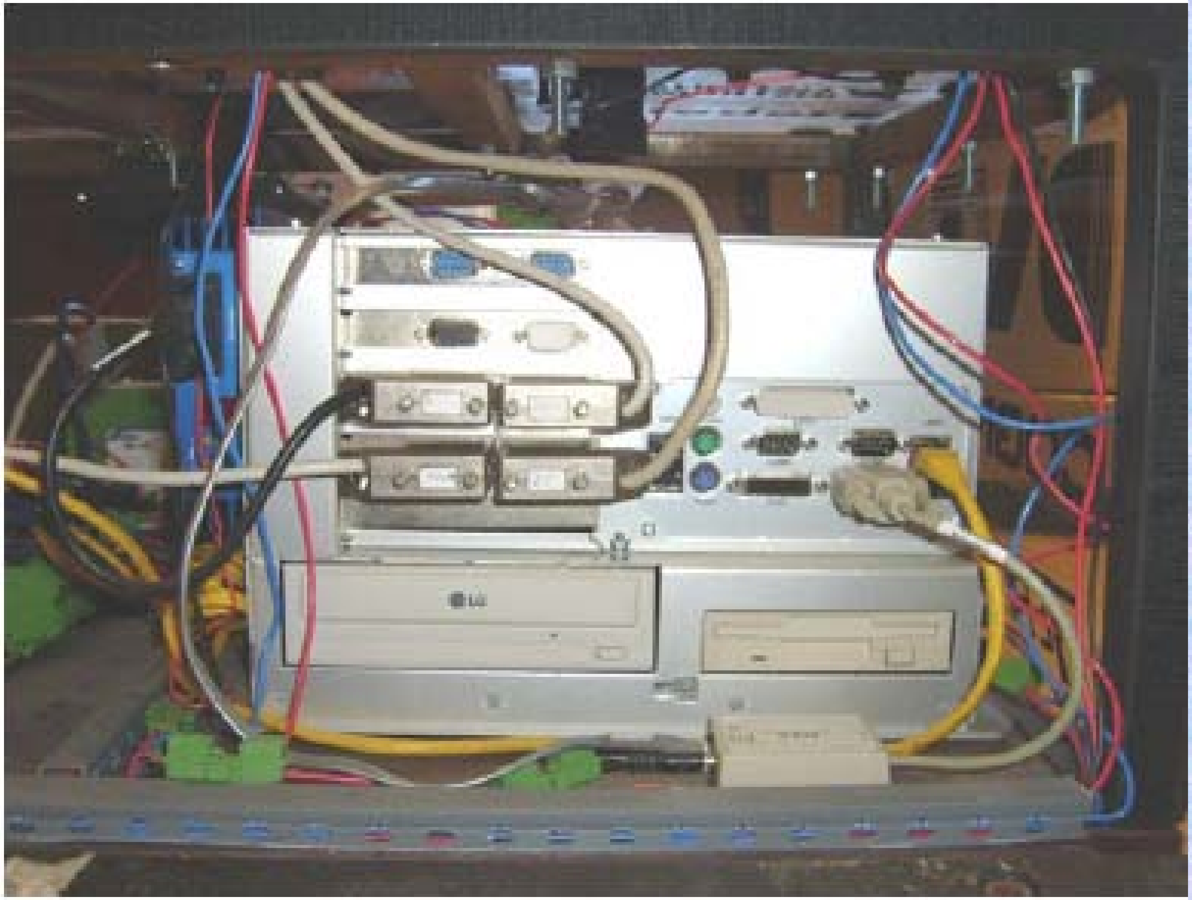


Fig. 2. Phoenix Contact IPC5500 Industrial PC, fully connected.

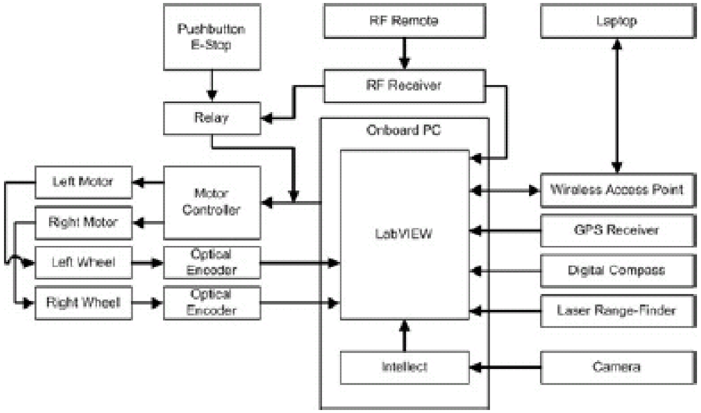


Fig. 8. Block diagram of Wunderbot IV subsystems.

SOURCE: Painter, J. and Wunderlich, J.T. (2008). **Wunderbot IV: autonomous robot for international competition**. In *Proceedings of the 12th World Multi-Conference on Systemics, Cybernetics and Informatics: WMSCI 2008, Orlando, FL*: (pp. 62-67). And HERE

Wunderbot IV

Fig. 3. Power wires running at the heart of the electrical system, from connectivity blocks (green) and fuses (black) mounted on a DIN rail, and then neatly tucked away in mounted plastic conduits. Phoenix Contact RAD-80211-XD wireless access point mounted on far left of rail.

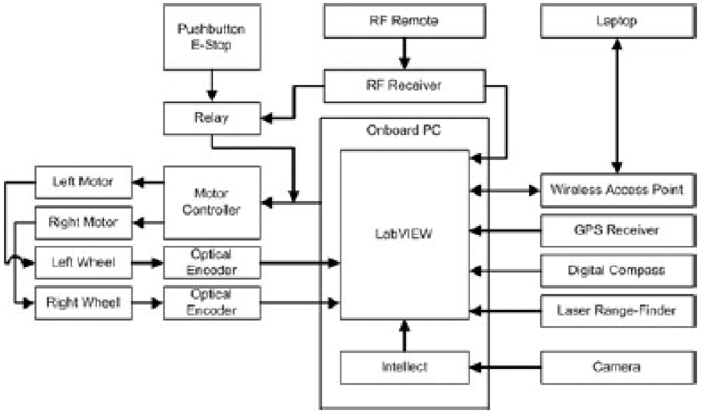


Fig. 8. Block diagram of Wunderbot IV subsystems.

SOURCE: Painter, J. and Wunderlich, J.T. (2008). **Wunderbot IV: autonomous robot for international competition.** In *Proceedings of the 12th World Multi-Conference on Systemics, Cybernetics and Informatics: WMSCI 2008, Orlando, FL*: (pp. 62-67). And HERE

Wunderbot IV

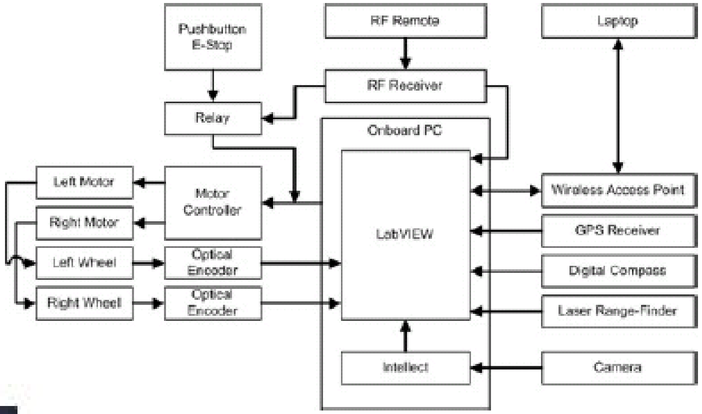


Fig. 8. Block diagram of Wunderbot IV subsystems.

(a) (b)

Fig. 4. (a) Spektrum DX6 model airplane remote control. (b) Wired pushbutton emergency stop and wireless emergency stop receiver.

SOURCE: Painter, J. and Wunderlich, J.T. (2008). **Wunderbot IV: autonomous robot for international competition**. In *Proceedings of the 12th World Multi-Conference on Systemics, Cybernetics and Informatics: WMSCI 2008, Orlando, FL*: (pp. 62-67). And HERE

Wunderbot IV

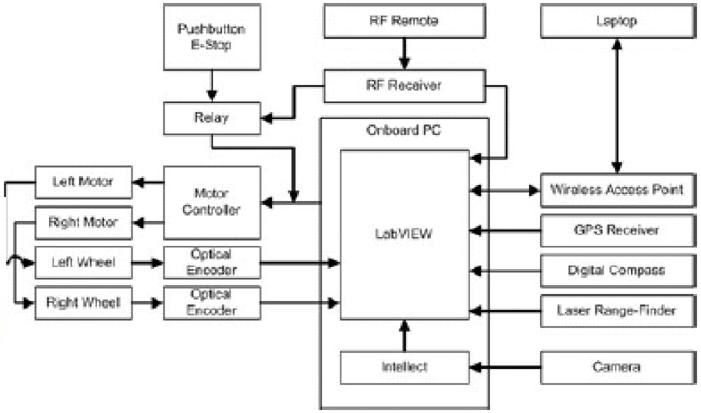


Fig. 8. Block diagram of Wunderbot IV subsystems.

(a) (b)

Fig. 5. (a) One of two SICK LMS 200 laser range-finders. (b) Cognex DVT Legend 554C XE camera.

Wunderbot 4

SOURCE: Painter, J. and Wunderlich, J.T. (2008). **Wunderbot IV: autonomous robot for international competition**. In *Proceedings of the 12th World Multi-Conference on Systemics, Cybernetics and Informatics: WMSCI 2008, Orlando, FL*: (pp. 62-67). And HERE

Wunderbot IV

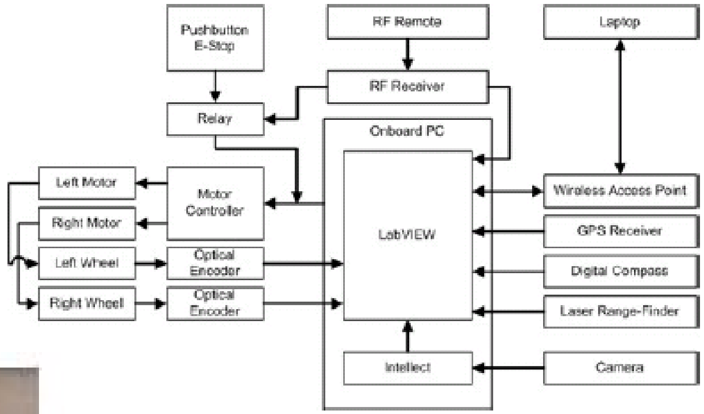
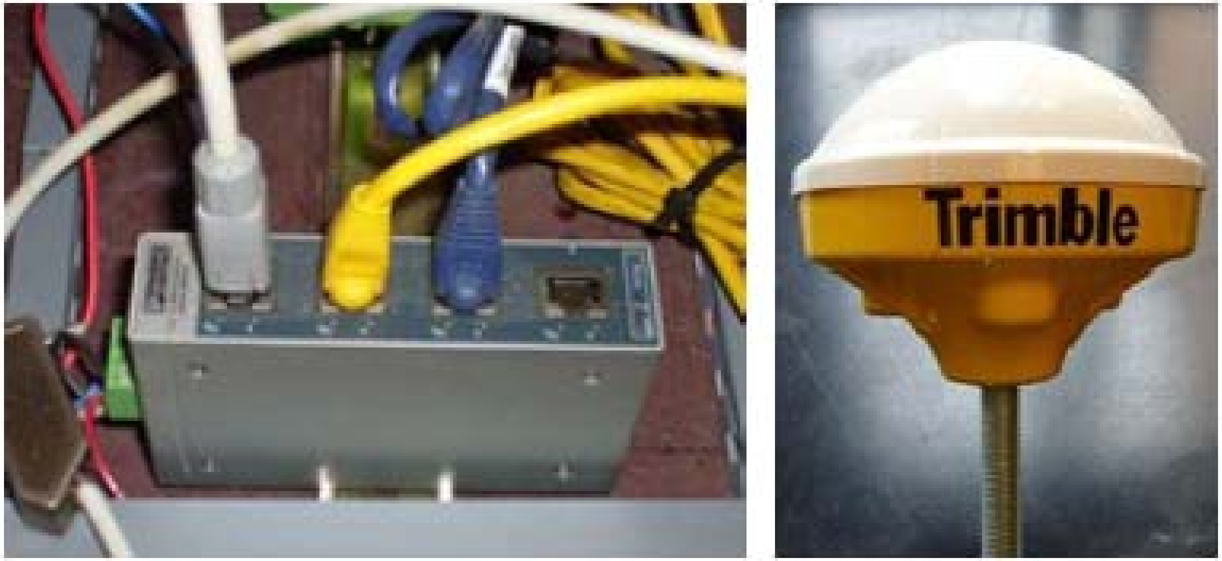


Fig. 8. Block diagram of Wunderbot IV subsystems.



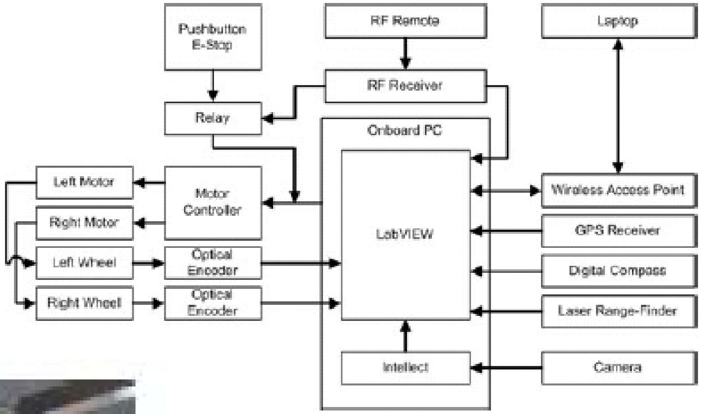
(a) (b)

Fig. 7. (a) Phoenix Contact SFN 5TX five-port switch, to which is connected, from left to right: camera, PC, wireless access point. (b) Trimble AgGPS 114.

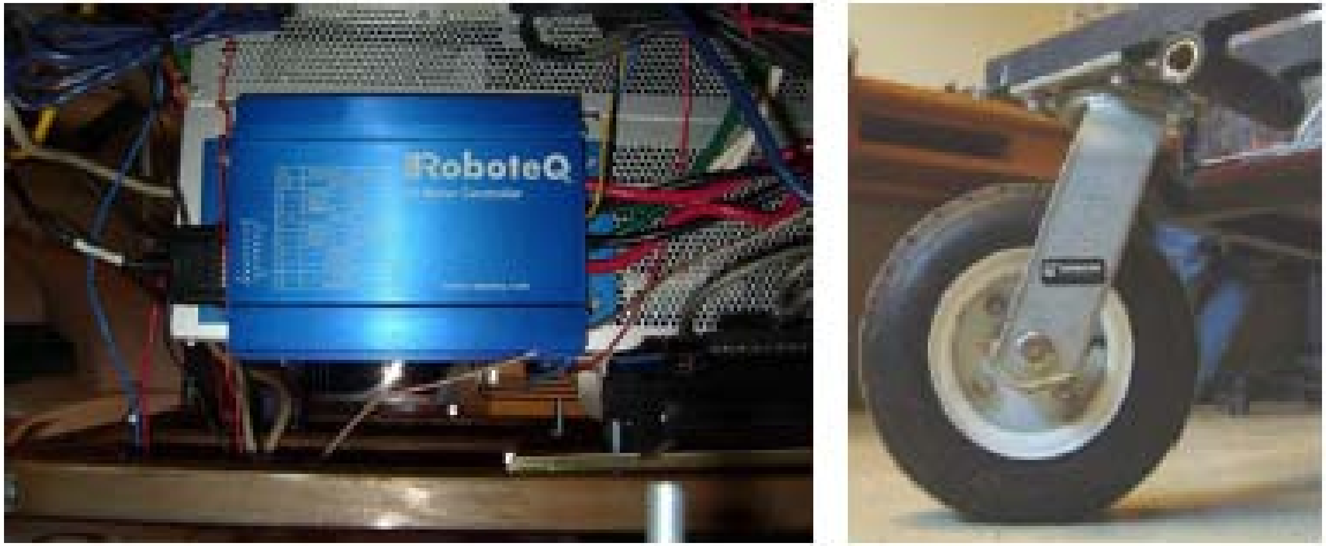
Wunderbot 4

SOURCE: Painter, J. and Wunderlich, J.T. (2008). **Wunderbot IV: autonomous robot for international competition**. In *Proceedings of the 12th World Multi-Conference on Systemics, Cybernetics and Informatics: WMSCI 2008, Orlando, FL*: (pp. 62-67). And HERE

Wunderbot IV



Block diagram of Wunderbot IV subsystems.



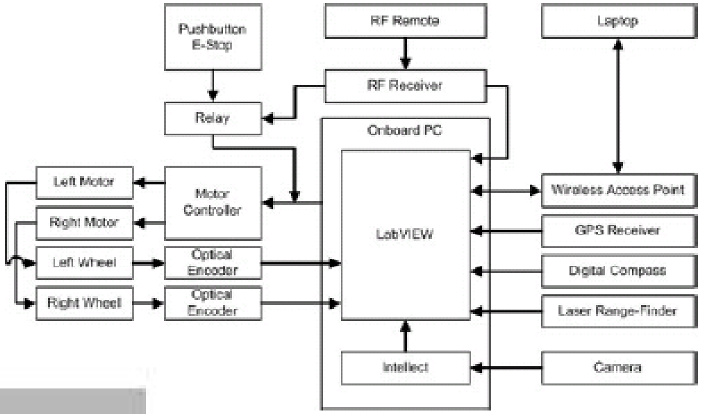
(a) (b)

Fig. 6. (a) Roboteq AX2550 motor controller. (b) One of two Hamilton Series 5000 pneumatic casters.

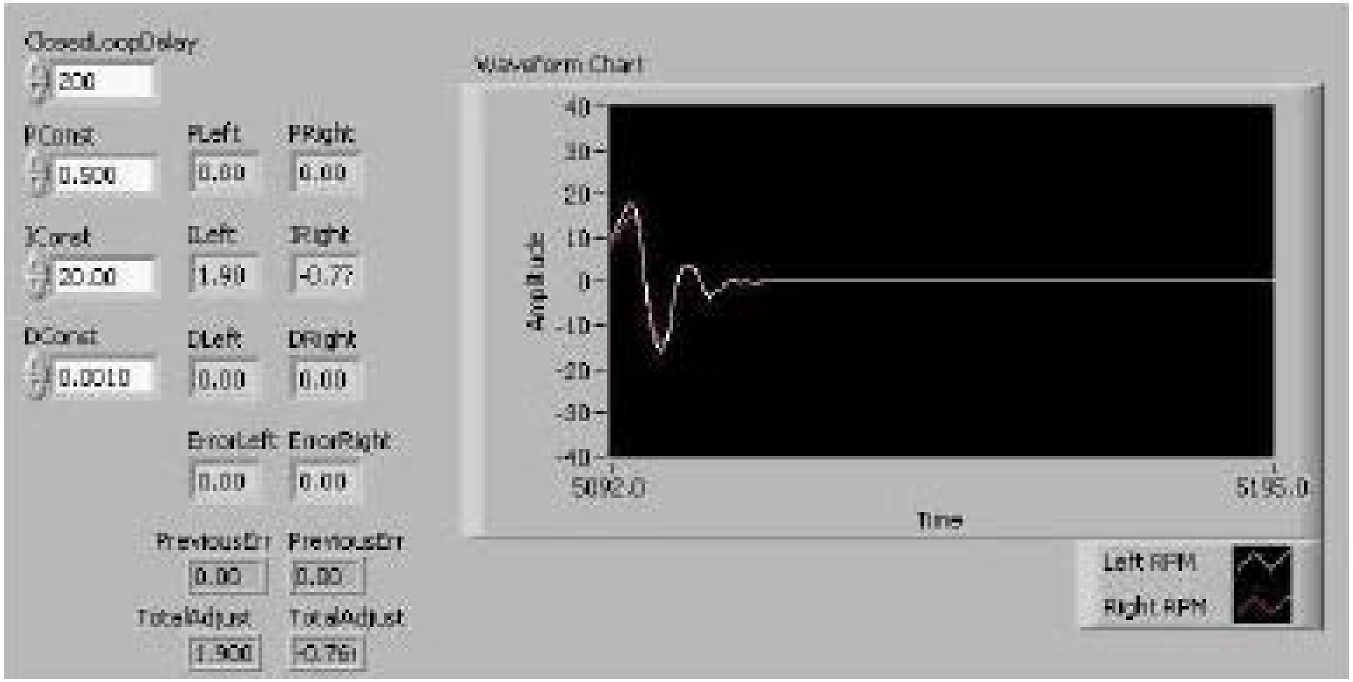
Wunderbot 4

SOURCE: Painter, J. and Wunderlich, J.T. (2008). **Wunderbot IV: autonomous robot for international competition**. In *Proceedings of the 12th World Multi-Conference on Systemics, Cybernetics and Informatics: WMSCI 2008, Orlando, FL*: (pp. 62-67). And HERE

Wunderbot IV



Block diagram of Wunderbot IV subsystems.



ClosedLoopDelay 200

PConst 0.500
PLeft 0.00
PRight 0.00

IConst 20.00
ILeft 1.90
IRight -0.77

DConst 0.0010
DLeft 0.00
DRight 0.00

ErrorLeft 0.00
ErrorRight 0.00

PreviousErr 0.00
PreviousErr 0.00

TotalAdjust 1.900
TotalAdjust -0.761

Fig. 5. LabVIEW control panel for PID controller with robot's resulting transient response.

SOURCE: Painter, J. G. (2008). **Vision system for Wunderbot IV autonomous robot**. *Elizabethtown College research report.*

Wunderbot 4 Wireless Communication

by Jeremy Crouse (advisor: J. Wunderlich)

Navigation

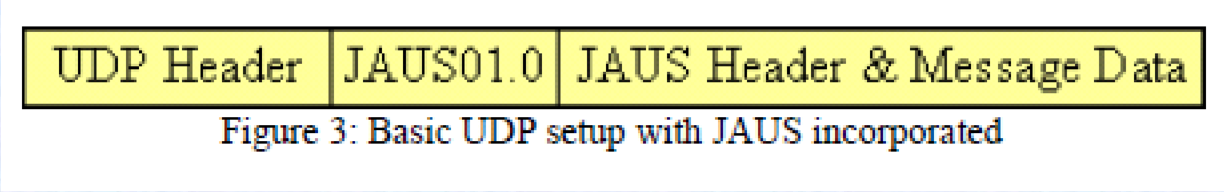
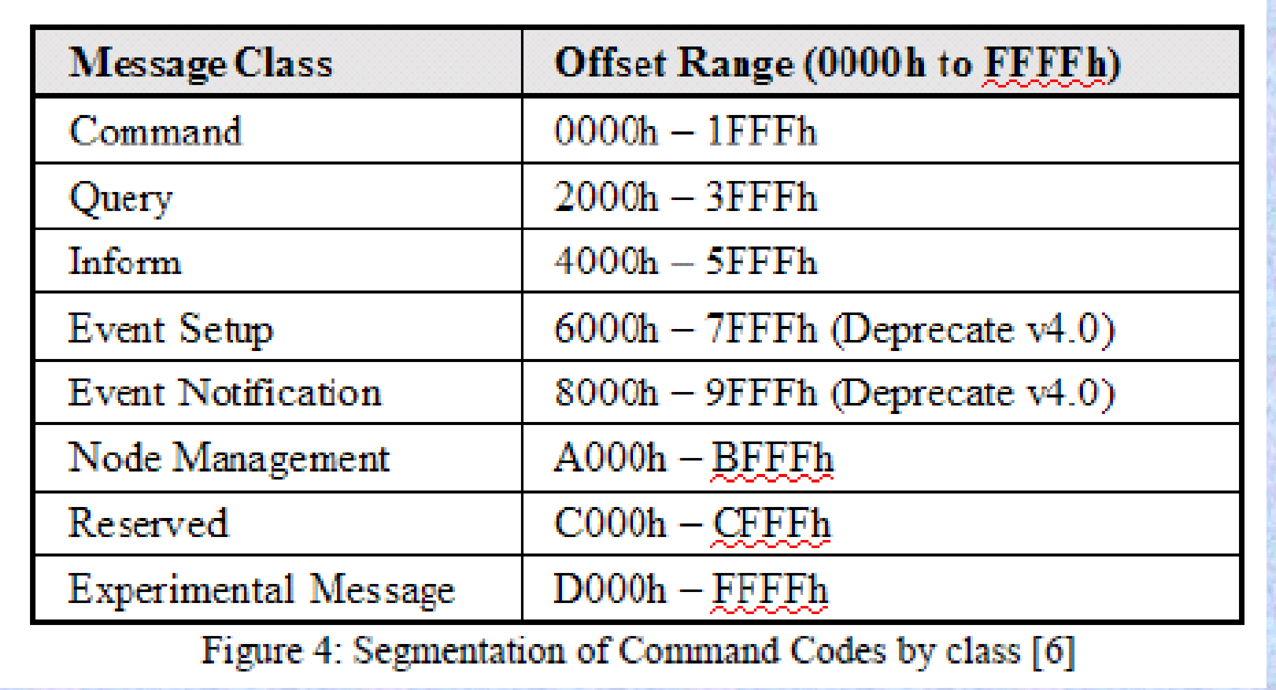


Figure 3: Basic UDP setup with JAUS incorporated



Message Class	Offset Range (0000h to FFFFh)
Command	0000h – 1FFFh
Query	2000h – 3FFFh
Inform	4000h – 5FFFh
Event Setup	6000h – 7FFFh (Deprecate v4.0)
Event Notification	8000h – 9FFFh (Deprecate v4.0)
Node Management	A000h – BFFFh
Reserved	C000h – CFFFh
Experimental Message	D000h – FFFFh

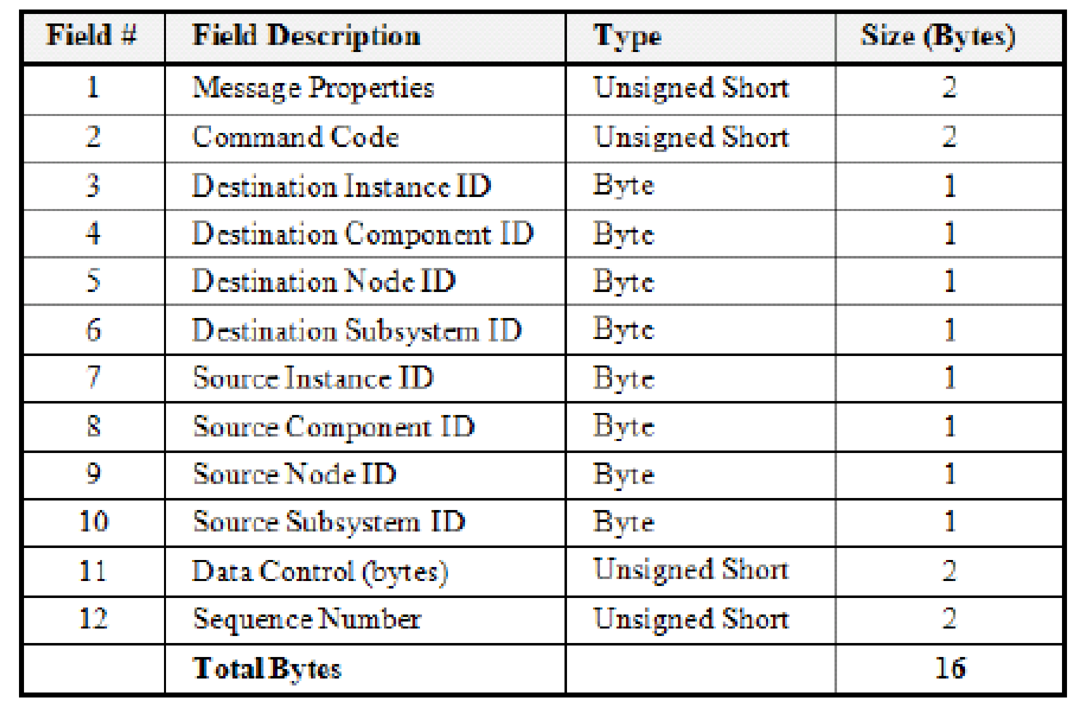
Figure 4: Segmentation of Command Codes by class [6]

Although Wunderbots are fully autonomous, the IGVC awards those who can respond to "JAUS"

SOURCE: : Crouse, J. (2008). **The joint architecture for unmanned systems: a subsystem of the wunderbot 4**. *Elizabethtown College research report.*

Wunderbot 4 Wireless Communication
by Jeremy Crouse (advisor: J. Wunderlich)

Navigation



Field #	Field Description	Type	Size (Bytes)
1	Message Properties	Unsigned Short	2
2	Command Code	Unsigned Short	2
3	Destination Instance ID	Byte	1
4	Destination Component ID	Byte	1
5	Destination Node ID	Byte	1
6	Destination Subsystem ID	Byte	1
7	Source Instance ID	Byte	1
8	Source Component ID	Byte	1
9	Source Node ID	Byte	1
10	Source Subsystem ID	Byte	1
11	Data Control (bytes)	Unsigned Short	2
12	Sequence Number	Unsigned Short	2
	Total Bytes		16

Figure 5: JAUS message header data format in bytes [6]

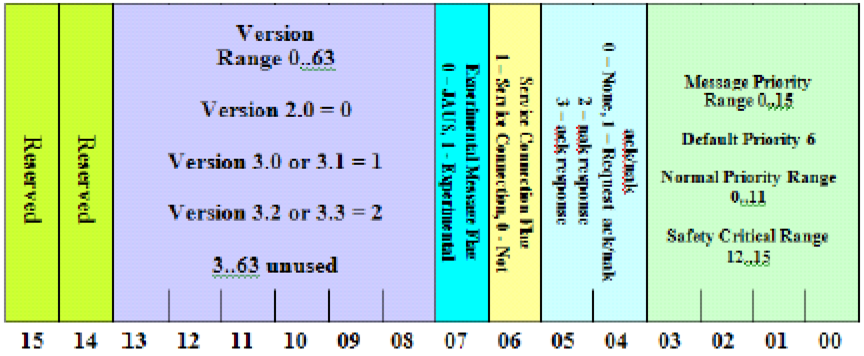


Figure 6: Message Property detailed structure [6]

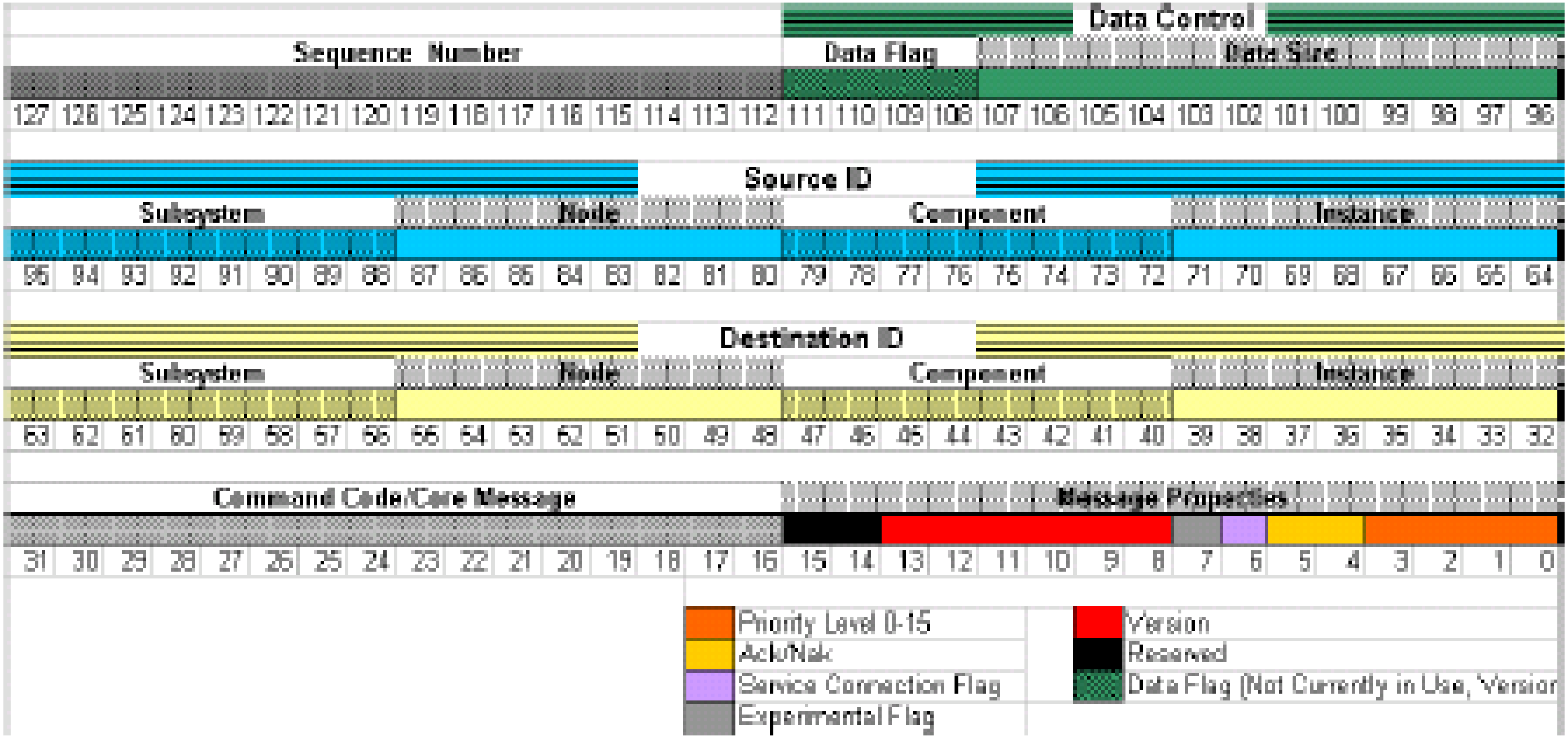
Although Wunderbots are fully autonomous, the IGVC awards those who can respond to “JAUS”

SOURCE: : Crouse, J. (2008). **The joint architecture for unmanned systems: a subsystem of the wunderbot 4**. *Elizabethtown College research report.*

Wunderbot 4 Wireless Communication
by Jeremy Crouse (advisor: J. Wunderlich)

Navigation

Although Wunderbots are fully autonomous, the IGVC awards those who can respond to "JAUS"


Sequence Number | Data Control: Data Flag, Data Size
127 126 125 124 123 122 121 120 119 118 117 116 115 114 113 112 111 110 109 108 107 106 105 104 103 102 101 100 99 98 97 96

Source ID: Subsystem, Node, Component, Instance
95 94 93 92 91 90 89 88 87 86 85 84 83 82 81 80 79 78 77 76 75 74 73 72 71 70 69 68 67 66 65 64

Destination ID: Subsystem, Node, Component, Instance
63 62 61 60 59 58 57 56 55 54 53 52 51 50 49 48 47 46 45 44 43 42 41 40 39 38 37 36 35 34 33 32

Command Code/Core Message | Message Properties
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

🟧	Priority Level 0-15	🟥	Version
🟨	Ack/Nak	⬛	Reserved
🟪	Service Connection Flag	▨	Data Flag (Not Currently in Use, Versio
▩	Experimental Flag		

Figure 7: JAUS message header detailed structure [6]

SOURCE: : Crouse, J. (2008). **The joint architecture for unmanned systems: a subsystem of the wunderbot 4**. *Elizabethtown College research report.*

See the LabVIEW computer implementation and integration of the most recent Wunderbot systems here:

Wunderbot - Main VI Labview Tutorial
Wunderbot - GPS Subsystem Labview Tutorial
Wunderbot - LADAR Subsystem Labview Tutorial
Wunderbot - JAUS Subsystem Labview Tutorial
Wunderbot - Vision Subsystem Labview Tutorial
Wunderbot - Motor Control Subsystem Labview Tutorial
Wunderbot - Digital Compass Subsystem Labview Tutorial
Wunderbot - MCglobal08 Subsystem Labview Tutorial
nanoLC Robot Simulation


Wunderbot 4

And computer hardware and software decisions here:

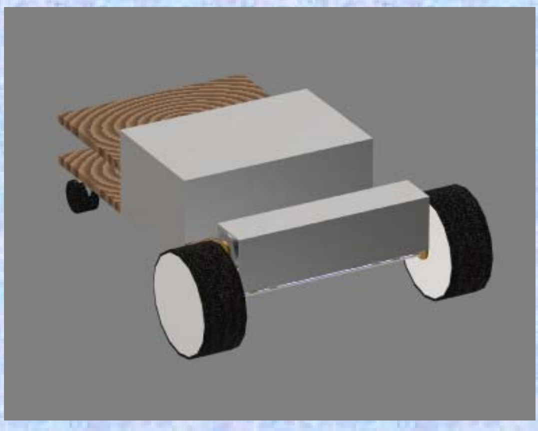
[1] Painter, J. and Wunderlich, J.T. (2008). **Wunderbot IV: autonomous robot for international competition**. In *Proceedings of the 12th World Multi-Conference on Systemics, Cybernetics and Informatics: WMSCI 2008, Orlando, FL*: (pp. 62-67). And HERE

[2] Coleman, D. and Wunderlich, J.T. (2008). **O^3: an optimal and opportunistic path planner (with obstacle avoidance) using voronoi polygons**. In *Proceedings of IEEE the 10th international Workshop on Advanced Motion Control, Trento, Italy.* vol. 1, (pp. 371-376). IEEE Press.

[3] JAUS wireless packetized communication by Jeremy Crouse

And the computer hardware and software schematics for past Wunderbots at these links:

MultEbot 1, 2001
“Wunderbot” 0



Wunderbot 0 website:
http://users.etown.edu/w/wunderjt/home_wunderbot0.html

Wunderbot 1
“MultEbot 2”

Wunderbot 1 website:
http://users.etown.edu/w/wunderjt/StudentProjects/Wunderbot%202003/Wunderbot%20Webpage2003/Robot%20webfiles/index.htm

NOTE: Students announced renaming of MultEbot 2 to “Wunderbot” at 2001 annual symposium (i.e., not Dr. Wunderlich’s idea)

Simulation vs. Real-Time control

Simulation	Real-Time control
Using good engineering and physics, create a model of a physical system (i.e., not just a cartoon)	Establish stable closed loop control with a good model (“Plant”) that represents physical system being controlled
Vary inputs to simulation to better understand model	Fine tune PID control to better manipulation of physical system
Use more complex computer hardware to enhance graphics and model complexity	**Intentionally simplify** all hardware to yield fast, compact, fault-tolerant, real-time responses
Use more complex computer software to enhance graphics and minimize programming effort	**Intentionally simplify** code to yield fast, compact, fault-tolerant, real-time responses. No operating system or a real-time OS may be best
Interact with real-time code to improve physical model and build ENVIRONMENTAL MAPS	Interact with simulation to obtain GLOBAL PATH-PLANNING rather than Local

Read more on Simulation vs. Real-time code, and Local vs. Global path-planning at:

Carsen, A., Rankin, J., Fuguson, D., and Stentz, A. (2007). **Global path planning on board the mars exploration rovers**. *In Proceedings of the IEEE Aerospace Conference, 2007. IEEE Press. (available at http://marstech.jpl.nasa.gov/publications/z02_0102.pdf)*

Coleman, D. and Wunderlich, J.T. (2008). ***O^3: an optimal and opportunistic path planner (with obstacle avoidance) using voronoi polygons*.** *In Proceedings of IEEE the 10th international Workshop on Advanced Motion Control, Trento, Italy. vol. 1, (pp. 371-376). IEEE Press.*

Campos, D. and Wunderlich, J. T. (2002). ***Development of an interactive simulation with real-time robots for search and rescue*.** *In Proceedings of IEEE/ASME International conference on Flexible Automation, Hiroshima, Japan: (session U-007). ASME Press.*

Wunderlich, J.T. (2001). ***Simulation vs. real-time control; with applications to robotics and neural networks*.** *In Proceedings of 2001 ASEE Annual Conference & Exposition, Albuquerque, NM: (session 2793), [CD-ROM]. ASEE Publications.*

Quality Assurance in Computer Design

- Functional verification on simulated prototype machine
- Digital & analog VLSI circuit simulation testing
- Functional verification on VLSI circuit simulation
- Functional verification on prototype hardware
- Instruction-mix and performance benchmark testing

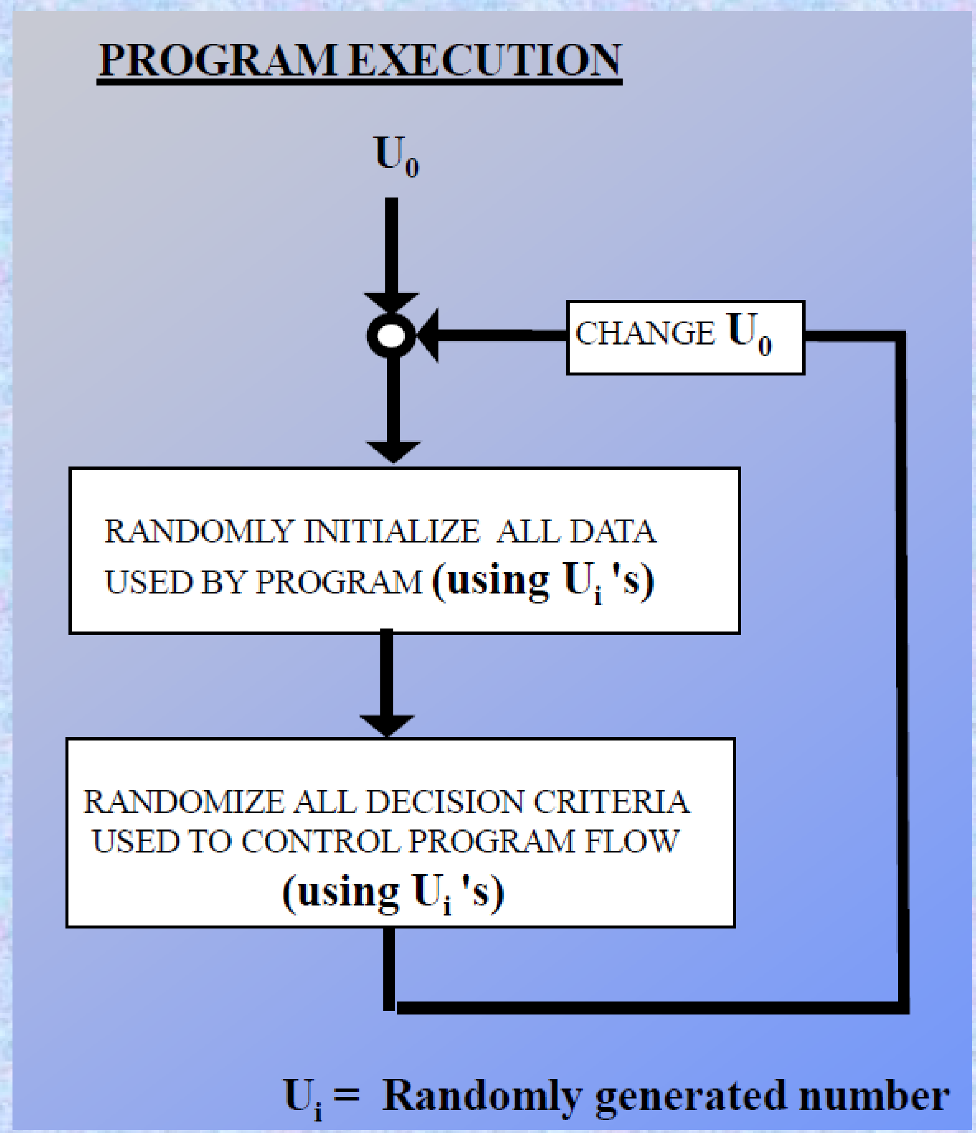
J. Wunderlich was a researcher at IBM before joining Purdue University as an Assistant Professor

His IBM research was on quality control of S/390 Multi-processor SMP supercomputers

See more here:

http://users.etown.edu/w/wunderjt/home_IBM.html

Quality Assurance in Computer Design



PROGRAM EXECUTION

U_0

CHANGE U_0

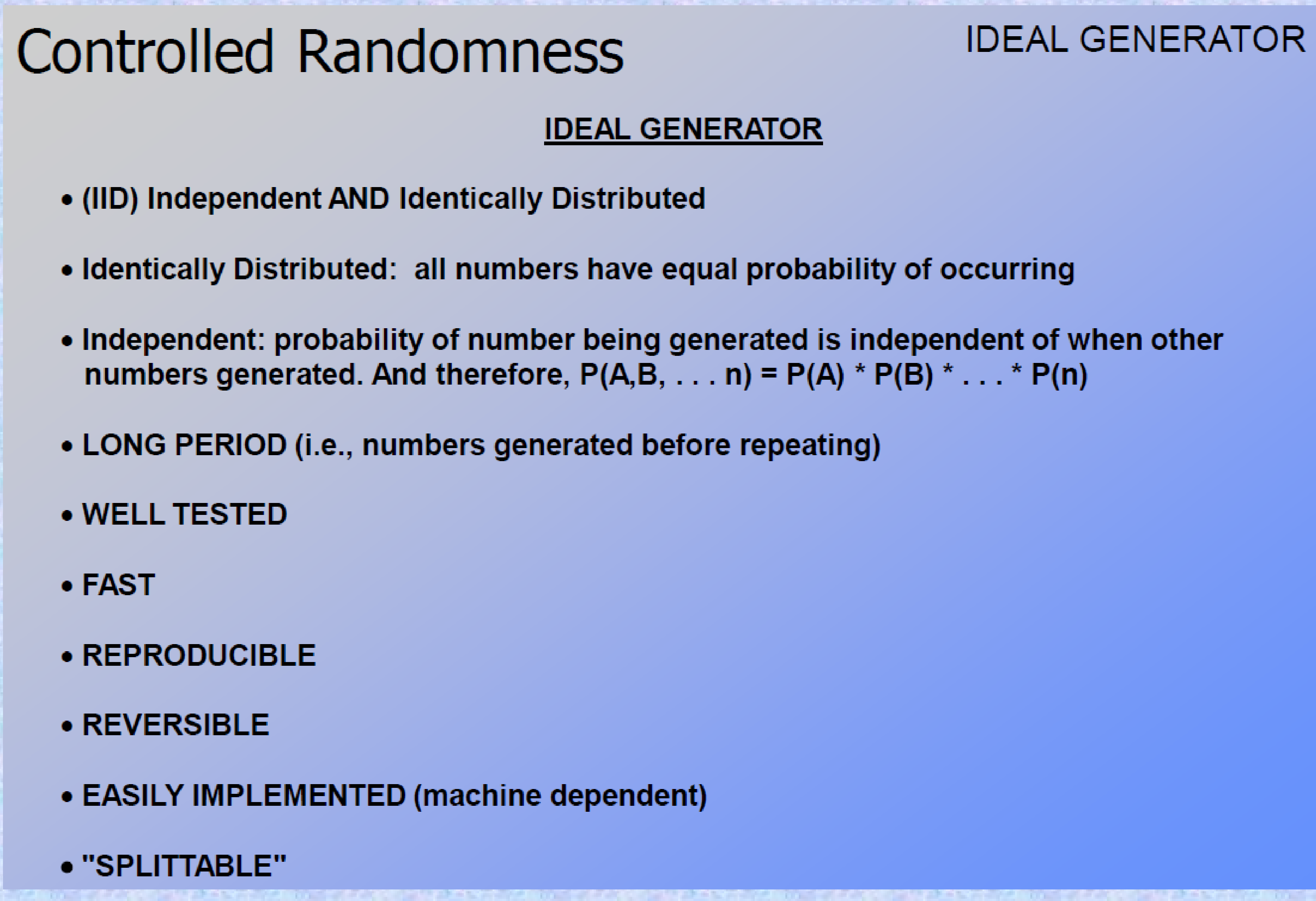
RANDOMLY INITIALIZE ALL DATA USED BY PROGRAM **(using U_i's)**

RANDOMIZE ALL DECISION CRITERIA USED TO CONTROL PROGRAM FLOW **(using U_i's)**

U_i = Randomly generated number

FROM: Wunderlich, J.T. (2003). **Functional verification of SMP, MPP, and vector-register supercomputers through controlled randomness**. In *Proceedings of IEEE SoutheastCon, Ocho Rios, Jamaica,* M. Curtis (Ed.): (pp. 117-122). IEEE Press.

and: Wunderlich, J.T. (1997). **Random number generator macros for the system assurance kernel product assurance macro interface**. Systems Programmer's User Manual for IBM S/390 Systems Architecture Verification, Poughkeepsie, NY.



Controlled Randomness

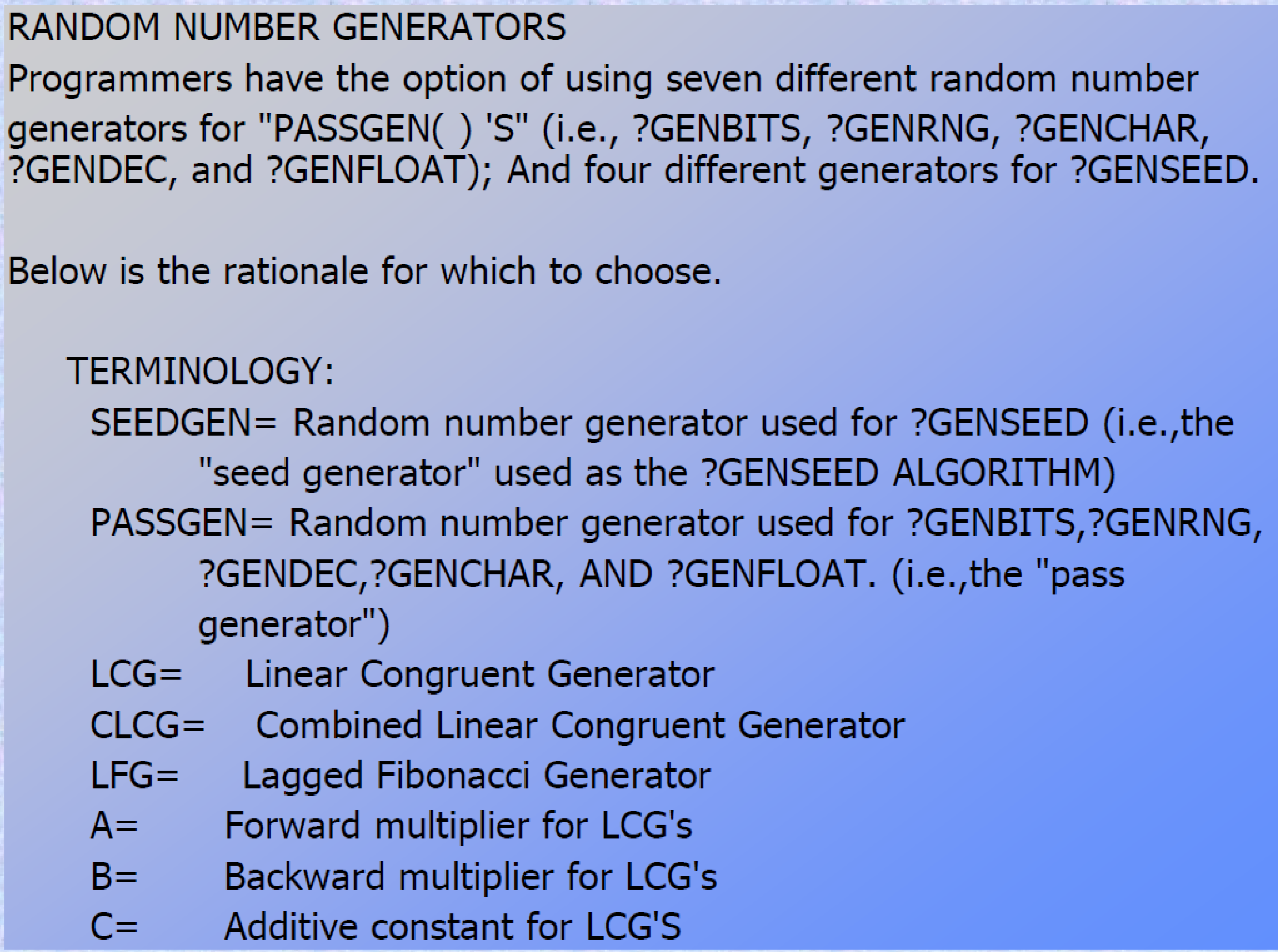
IDEAL GENERATOR

IDEAL GENERATOR

- (IID) Independent AND Identically Distributed
- Identically Distributed: all numbers have equal probability of occurring
- Independent: probability of number being generated is independent of when other numbers generated. And therefore, P(A,B, . . . n) = P(A) * P(B) * . . . * P(n)
- LONG PERIOD (i.e., numbers generated before repeating)
- WELL TESTED
- FAST
- REPRODUCIBLE
- REVERSIBLE
- EASILY IMPLEMENTED (machine dependent)
- "SPLITTABLE"

FROM: Wunderlich, J.T. (2003). **Functional verification of SMP, MPP, and vector-register supercomputers through controlled randomness**. In *Proceedings of IEEE SoutheastCon, Ocho Rios, Jamaica,* M. Curtis (Ed.): (pp. 117-122). IEEE Press.

and: Wunderlich, J.T. (1997). **Random number generator macros for the system assurance kernel product assurance macro interface**. Systems Programmer's User Manual for IBM S/390 Systems Architecture Verification, Poughkeepsie, NY.



RANDOM NUMBER GENERATORS

Programmers have the option of using seven different random number generators for "PASSGEN( ) 'S" (i.e., ?GENBITS, ?GENRNG, ?GENCHAR, ?GENDEC, and ?GENFLOAT); And four different generators for ?GENSEED.

Below is the rationale for which to choose.

TERMINOLOGY:

SEEDGEN= Random number generator used for ?GENSEED (i.e.,the "seed generator" used as the ?GENSEED ALGORITHM)

PASSGEN= Random number generator used for ?GENBITS,?GENRNG, ?GENDEC,?GENCHAR, AND ?GENFLOAT. (i.e.,the "pass generator")

LCG= Linear Congruent Generator

CLCG= Combined Linear Congruent Generator

LFG= Lagged Fibonacci Generator

A= Forward multiplier for LCG's

B= Backward multiplier for LCG's

C= Additive constant for LCG'S

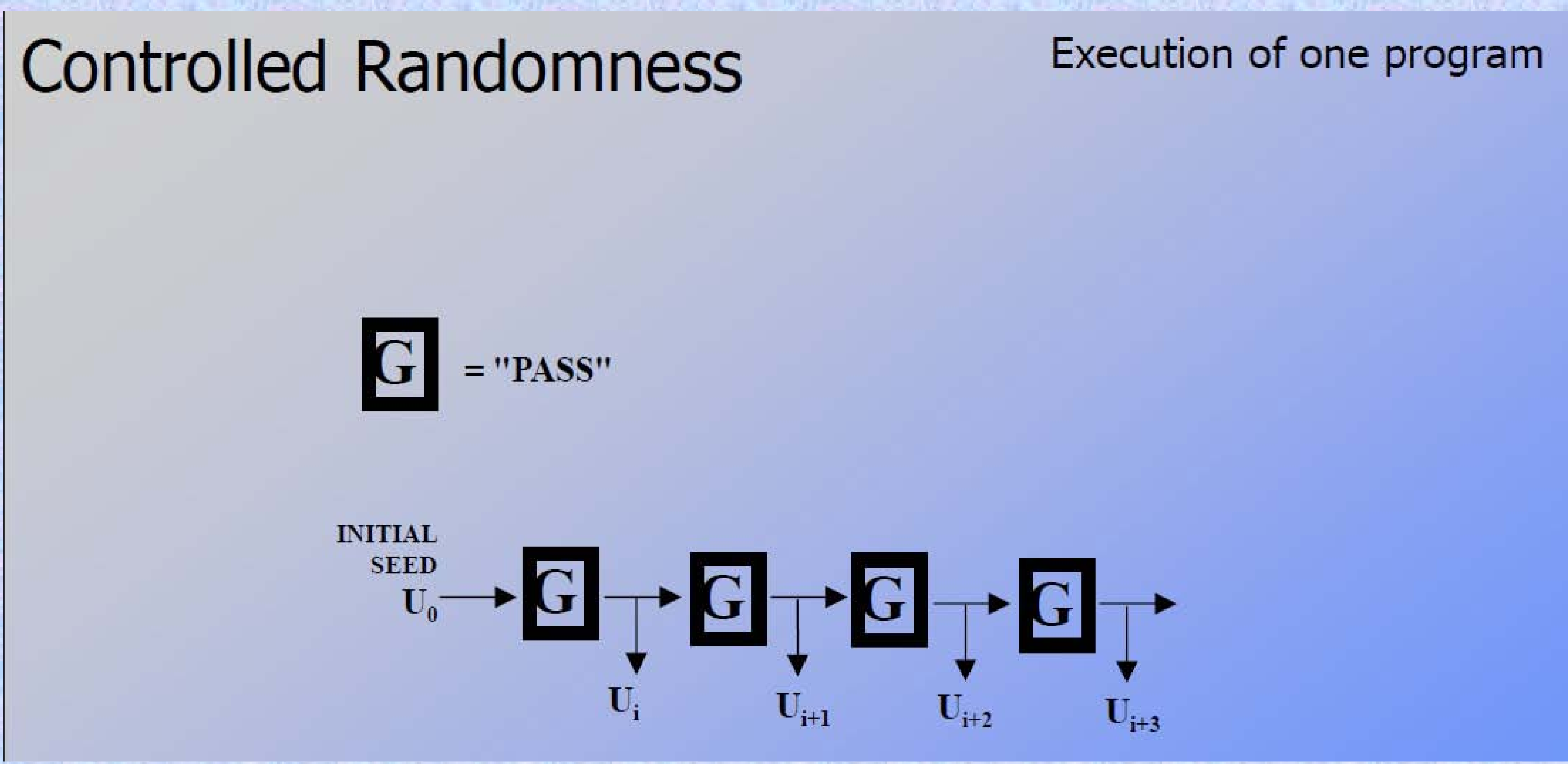
FROM: Wunderlich, J.T. (2003). **Functional verification of SMP, MPP, and vector-register supercomputers through controlled randomness**. In *Proceedings of IEEE SoutheastCon, Ocho Rios, Jamaica,* M. Curtis (Ed.): (pp. 117-122). IEEE Press.

and: Wunderlich, J.T. (1997). **Random number generator macros for the system assurance kernel product assurance macro interface**. Systems Programmer's User Manual for IBM S/390 Systems Architecture Verification, Poughkeepsie, NY.

X{I}= Present seed
X{I-1}= Previous seed
Q= Special "decomposition" variable for LCG's
R= Special "decomposition" variable for LCG's
M= Modulus
M_CLCG= Modulus for CLCG
J= Lag for LFG'S (the longer one)
K= Lag for LFG'S
X{I-J}= Previous {I-J} seed from LFG seed array
X{I-K}= Previous {I-K} seed from LFG seed array
OPERTR= The arithematic operator used for the LFG (+,OR *)
PERIOD= How many numbers generated before sequence repeats (i.e.,the cycle-length)

FROM: Wunderlich, J.T. (2003). **Functional verification of SMP, MPP, and vector-register supercomputers through controlled randomness**. In *Proceedings of IEEE SoutheastCon, Ocho Rios, Jamaica,* M. Curtis (Ed.): (pp. 117-122). IEEE Press.

and: Wunderlich, J.T. (1997). **Random number generator macros for the system assurance kernel product assurance macro interface**. Systems Programmer's User Manual for IBM S/390 Systems Architecture Verification, Poughkeepsie, NY.



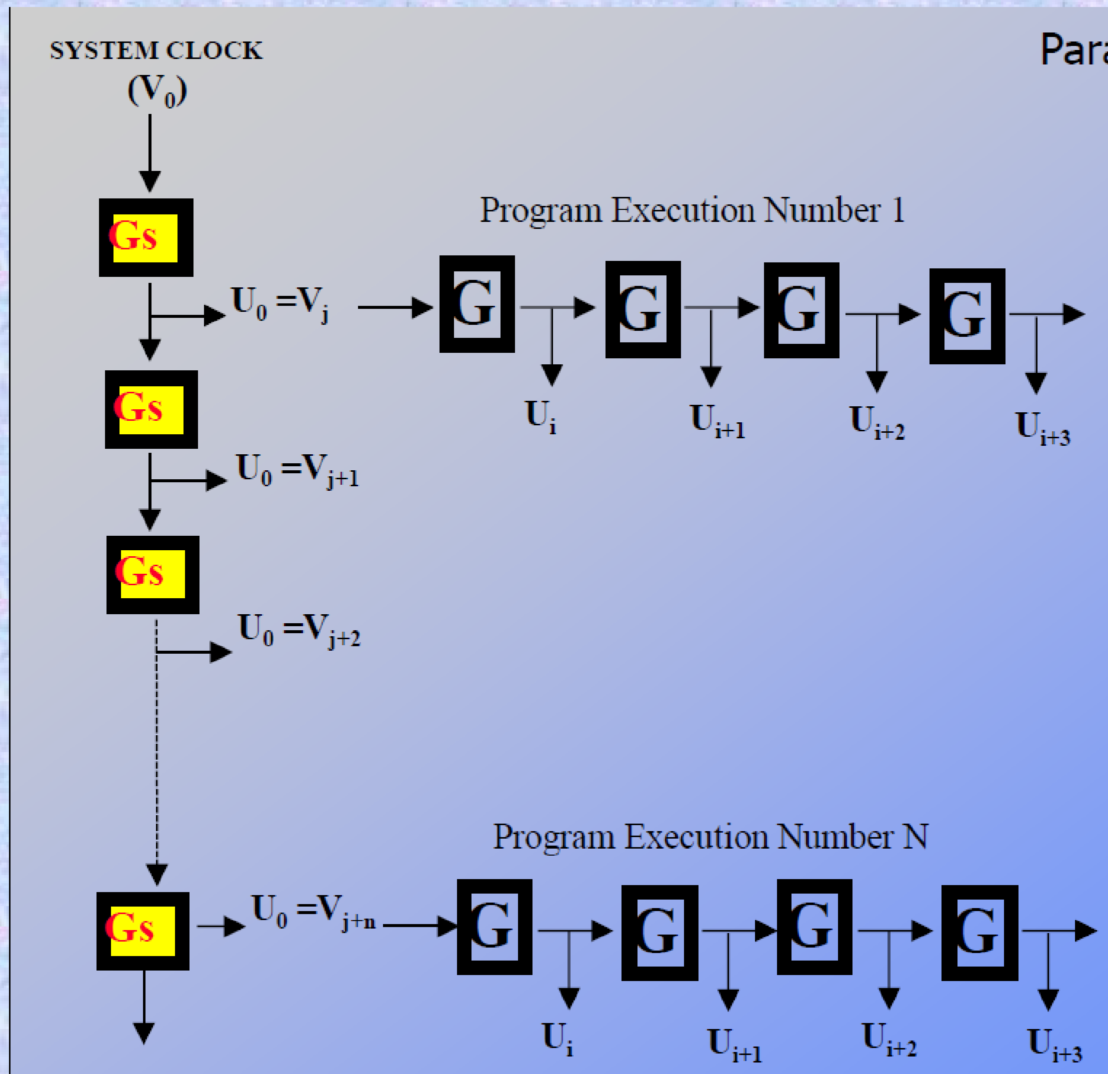
Controlled Randomness

Execution of one program

FROM: Wunderlich, J.T. (2003). **Functional verification of SMP, MPP, and vector-register supercomputers through controlled randomness**. In *Proceedings of IEEE SoutheastCon, Ocho Rios, Jamaica,* M. Curtis (Ed.): (pp. 117-122). IEEE Press.

and: Wunderlich, J.T. (1997). **Random number generator macros for the system assurance kernel product assurance macro interface**. Systems Programmer's User Manual for IBM S/390 Systems Architecture Verification, Poughkeepsie, NY.

Parallel Program Execution

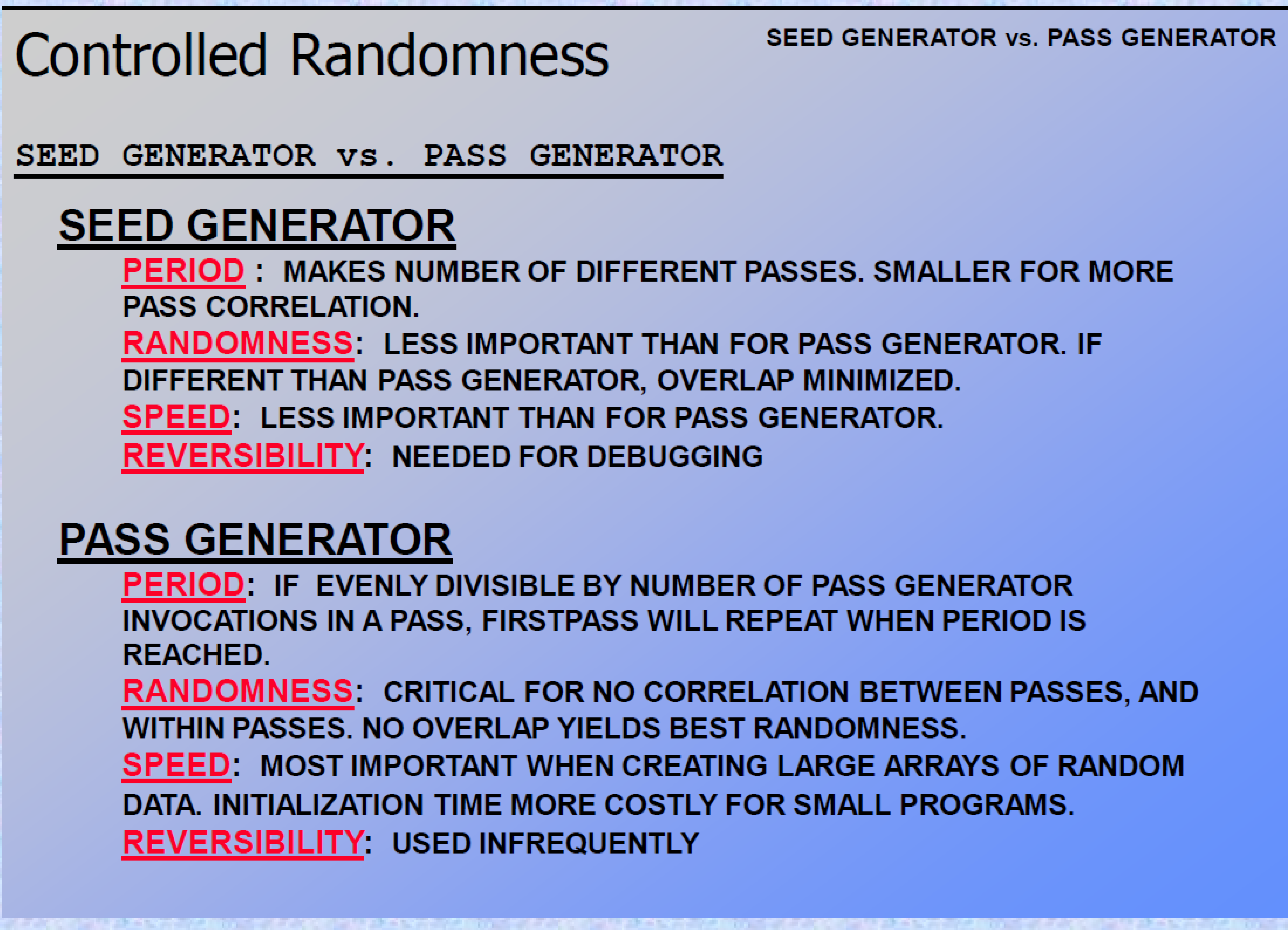


Gs = SEED GENERATOR

G = PASS GENERATOR

FROM: Wunderlich, J.T. (2003). Functional verification of SMP, MPP, and vector-register supercomputers through controlled randomness. In *Proceedings of IEEE SoutheastCon, Ocho Rios, Jamaica,* M. Curtis (Ed.): (pp. 117-122). IEEE Press.

and: Wunderlich, J.T. (1997). Random number generator macros for the system assurance kernel product assurance macro interface. Systems Programmer's User Manual for IBM S/390 Systems Architecture Verification, Poughkeepsie, NY.



SEED GENERATOR vs. PASS GENERATOR

Controlled Randomness

SEED GENERATOR vs. PASS GENERATOR

SEED GENERATOR

PERIOD : MAKES NUMBER OF DIFFERENT PASSES. SMALLER FOR MORE PASS CORRELATION.

RANDOMNESS: LESS IMPORTANT THAN FOR PASS GENERATOR. IF DIFFERENT THAN PASS GENERATOR, OVERLAP MINIMIZED.

SPEED: LESS IMPORTANT THAN FOR PASS GENERATOR.

REVERSIBILITY: NEEDED FOR DEBUGGING

PASS GENERATOR

PERIOD: IF EVENLY DIVISIBLE BY NUMBER OF PASS GENERATOR INVOCATIONS IN A PASS, FIRSTPASS WILL REPEAT WHEN PERIOD IS REACHED.

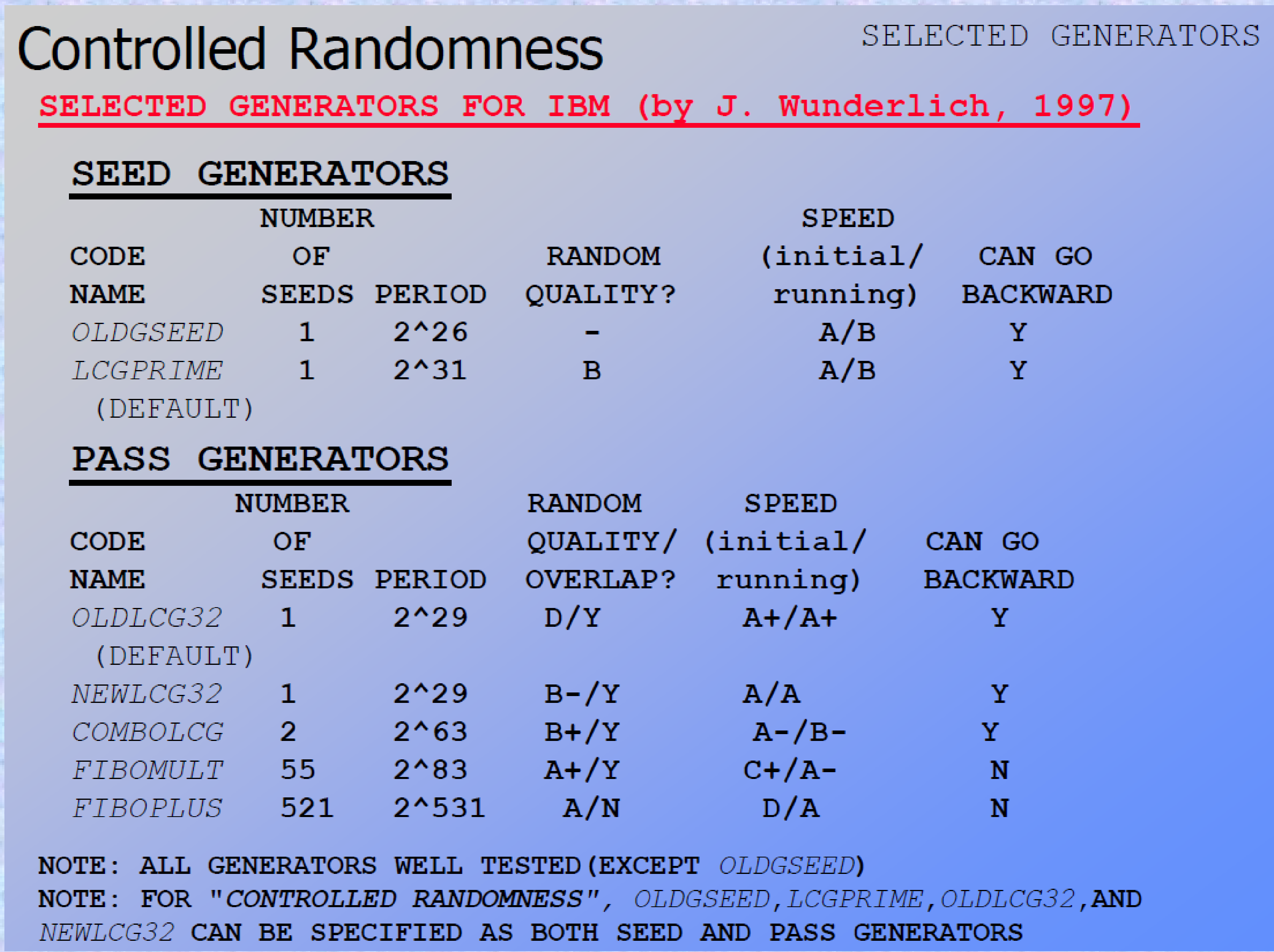
RANDOMNESS: CRITICAL FOR NO CORRELATION BETWEEN PASSES, AND WITHIN PASSES. NO OVERLAP YIELDS BEST RANDOMNESS.

SPEED: MOST IMPORTANT WHEN CREATING LARGE ARRAYS OF RANDOM DATA. INITIALIZATION TIME MORE COSTLY FOR SMALL PROGRAMS.

REVERSIBILITY: USED INFREQUENTLY

FROM: Wunderlich, J.T. (2003). **Functional verification of SMP, MPP, and vector-register supercomputers through controlled randomness**. In *Proceedings of IEEE SoutheastCon, Ocho Rios, Jamaica*, M. Curtis (Ed.): (pp. 117-122). IEEE Press.

and: Wunderlich, J.T. (1997). **Random number generator macros for the system assurance kernel product assurance macro interface**. Systems Programmer's User Manual for IBM S/390 Systems Architecture Verification, Poughkeepsie, NY.



Controlled Randomness

SELECTED GENERATORS

SELECTED GENERATORS FOR IBM (by J. Wunderlich, 1997)

SEED GENERATORS

CODE NAME	NUMBER OF SEEDS	PERIOD	RANDOM QUALITY?	SPEED (initial/ running)	CAN GO BACKWARD
OLDGSEED	1	2^26	-	A/B	Y
LCGPRIME (DEFAULT)	1	2^31	B	A/B	Y

PASS GENERATORS

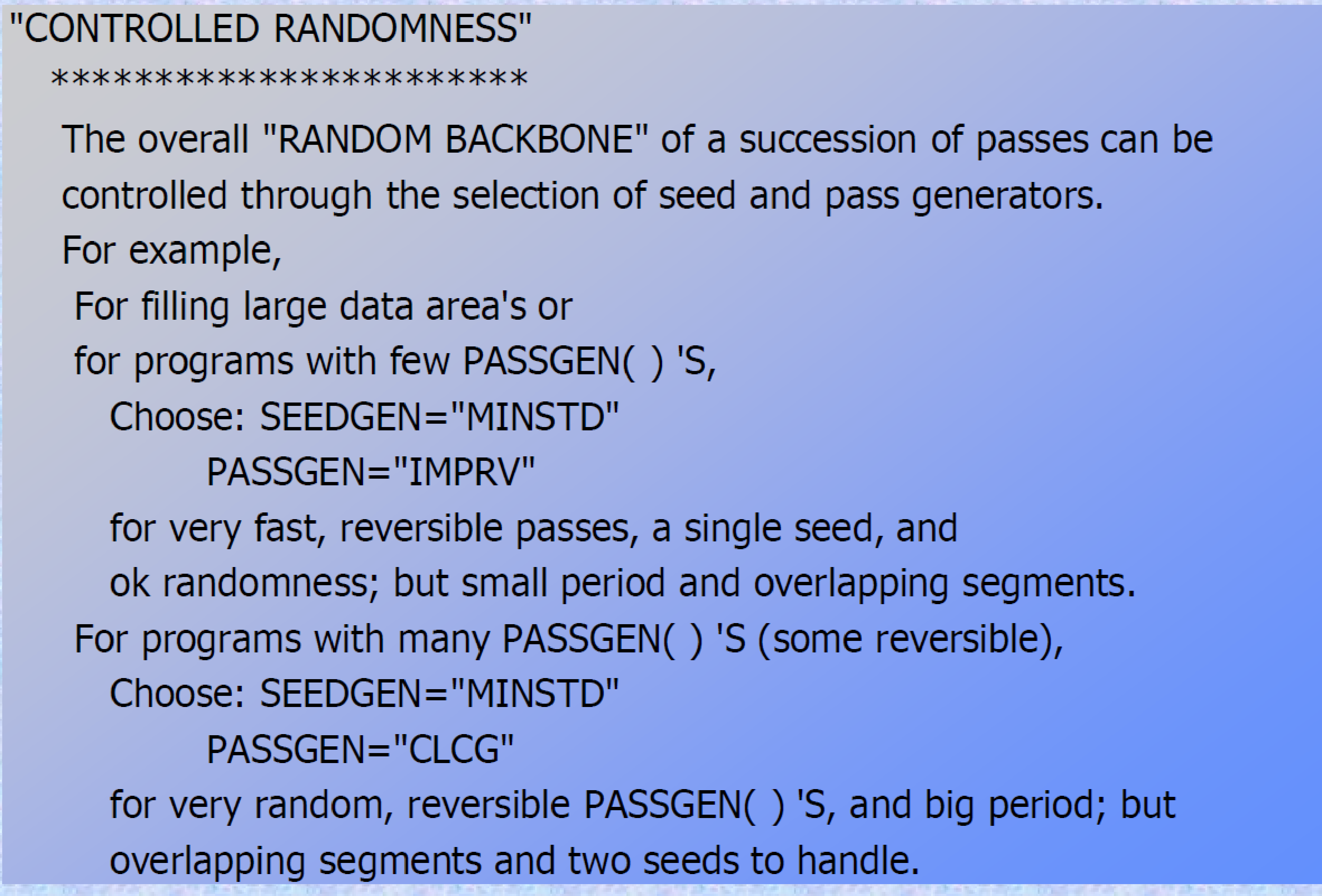
CODE NAME	NUMBER OF SEEDS	PERIOD	RANDOM QUALITY/ OVERLAP?	SPEED (initial/ running)	CAN GO BACKWARD
OLDLCG32 (DEFAULT)	1	2^29	D/Y	A+/A+	Y
NEWLCG32	1	2^29	B-/Y	A/A	Y
COMBOLCG	2	2^63	B+/Y	A-/B-	Y
FIBOMULT	55	2^83	A+/Y	C+/A-	N
FIBOPLUS	521	2^531	A/N	D/A	N

NOTE: ALL GENERATORS WELL TESTED (EXCEPT *OLDGSEED*)

NOTE: FOR "*CONTROLLED RANDOMNESS*", *OLDGSEED*, *LCGPRIME*, *OLDLCG32*, AND *NEWLCG32* CAN BE SPECIFIED AS BOTH SEED AND PASS GENERATORS

FROM: Wunderlich, J.T. (2003). **Functional verification of SMP, MPP, and vector-register supercomputers through controlled randomness**. In *Proceedings of IEEE SoutheastCon, Ocho Rios, Jamaica,* M. Curtis (Ed.): (pp. 117-122). IEEE Press.

and: Wunderlich, J.T. (1997). **Random number generator macros for the system assurance kernel product assurance macro interface**. Systems Programmer's User Manual for IBM S/390 Systems Architecture Verification, Poughkeepsie, NY.



"CONTROLLED RANDOMNESS"

****

The overall "RANDOM BACKBONE" of a succession of passes can be controlled through the selection of seed and pass generators.

For example,

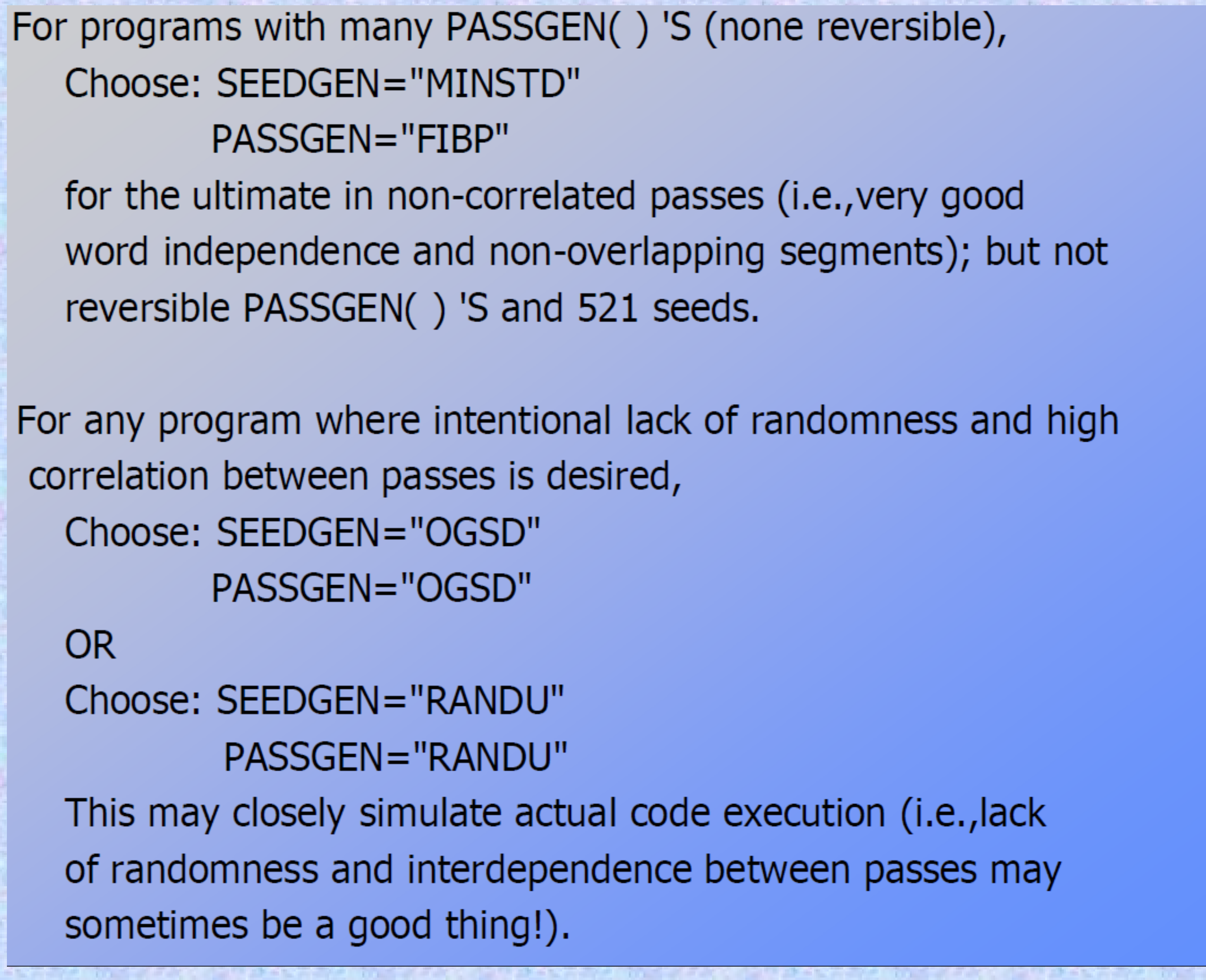
For filling large data area's or
for programs with few PASSGEN( ) 'S,
Choose: SEEDGEN="MINSTD"
PASSGEN="IMPRV"
for very fast, reversible passes, a single seed, and
ok randomness; but small period and overlapping segments.

For programs with many PASSGEN( ) 'S (some reversible),
Choose: SEEDGEN="MINSTD"
PASSGEN="CLCG"
for very random, reversible PASSGEN( ) 'S, and big period; but
overlapping segments and two seeds to handle.

FROM: Wunderlich, J.T. (2003). **Functional verification of SMP, MPP, and vector-register supercomputers through controlled randomness**. In *Proceedings of IEEE SoutheastCon, Ocho Rios, Jamaica,* M. Curtis (Ed.): (pp. 117-122). IEEE Press.

and: Wunderlich, J.T. (1997). **Random number generator macros for the system assurance kernel product assurance macro interface**. Systems Programmer's User Manual for IBM S/390 Systems Architecture Verification, Poughkeepsie, NY.

Quality Assurance in Computer Design



For programs with many PASSGEN( ) 'S (none reversible),

Choose: SEEDGEN="MINSTD"
PASSGEN="FIBP"

for the ultimate in non-correlated passes (i.e.,very good word independence and non-overlapping segments); but not reversible PASSGEN( ) 'S and 521 seeds.

For any program where intentional lack of randomness and high correlation between passes is desired,

Choose: SEEDGEN="OGSD"
PASSGEN="OGSD"

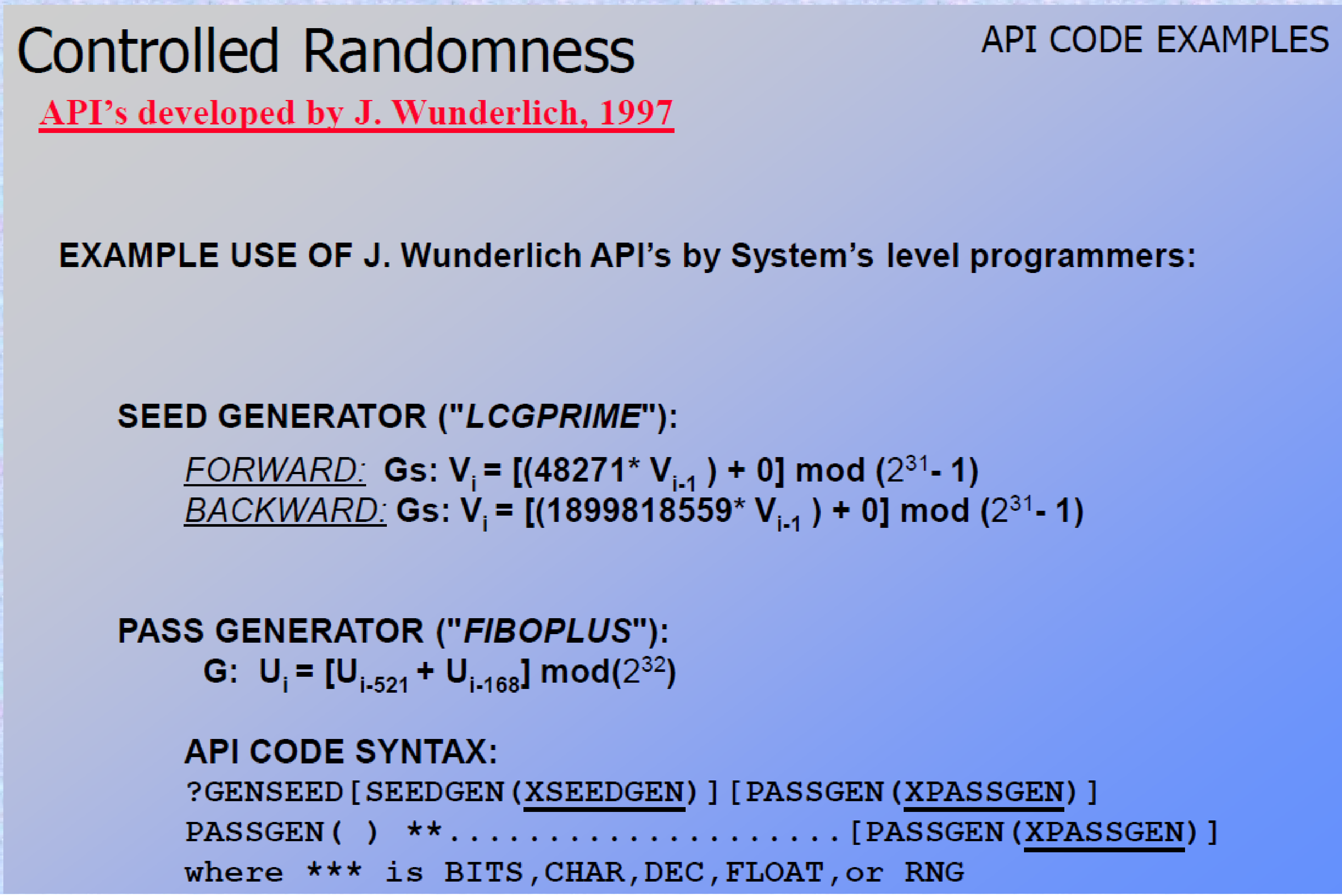
OR

Choose: SEEDGEN="RANDU"
PASSGEN="RANDU"

This may closely simulate actual code execution (i.e.,lack of randomness and interdependence between passes may sometimes be a good thing!).

FROM: Wunderlich, J.T. (2003). **Functional verification of SMP, MPP, and vector-register supercomputers through controlled randomness**. In *Proceedings of IEEE SoutheastCon, Ocho Rios, Jamaica,* M. Curtis (Ed.): (pp. 117-122). IEEE Press.

and: Wunderlich, J.T. (1997). **Random number generator macros for the system assurance kernel product assurance macro interface**. Systems Programmer's User Manual for IBM S/390 Systems Architecture Verification, Poughkeepsie, NY.



Controlled Randomness

API CODE EXAMPLES

API's developed by J. Wunderlich, 1997

EXAMPLE USE OF J. Wunderlich API's by System's level programmers:

SEED GENERATOR ("*LCGPRIME*"):

FORWARD: **Gs:** $V_i = [(48271 * V_{i-1}) + 0] \bmod (2^{31} - 1)$

BACKWARD: **Gs:** $V_i = [(1899818559 * V_{i-1}) + 0] \bmod (2^{31} - 1)$

PASS GENERATOR ("*FIBOPLUS*"):

G: $U_i = [U_{i-521} + U_{i-168}] \bmod (2^{32})$

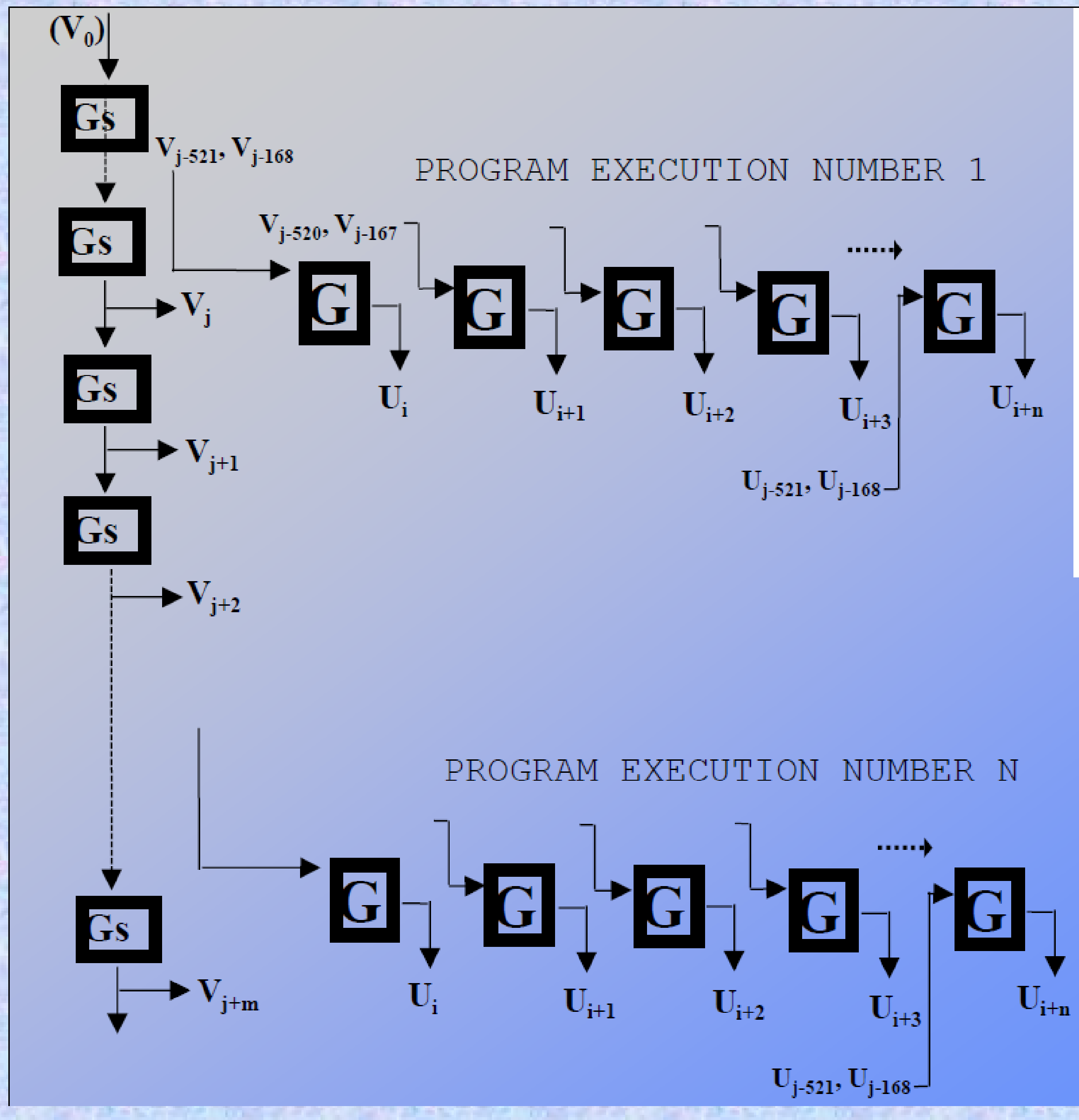
API CODE SYNTAX:

```
?GENSEED[SEEDGEN(XSEEDGEN)][PASSGEN(XPASSGEN)]
PASSGEN( ) **......................[PASSGEN(XPASSGEN)]
where *** is BITS,CHAR,DEC,FLOAT,or RNG
```

FROM: Wunderlich, J.T. (2003). **Functional verification of SMP, MPP, and vector-register supercomputers through controlled randomness**. In *Proceedings of IEEE SoutheastCon, Ocho Rios, Jamaica,* M. Curtis (Ed.): (pp. 117-122). IEEE Press.

and: Wunderlich, J.T. (1997). **Random number generator macros for the system assurance kernel product assurance macro interface**. Systems Programmer's User Manual for IBM S/390 Systems Architecture Verification, Poughkeepsie, NY.

Quality Assurance in Computer Design



API CODE EXAMPLE

SEED GENERATOR
("*LCGPRIME*"):

FORWARD:
Gs: $V_i = [(48271 * V_{i-1}) + 0] \bmod (2^{31}-1)$
BACKWARD:
Gs: $V_i = [(1899818559 * V_{i-1}) + 0] \bmod (2^{31}-1)$

PASS GENERATOR
("*FIBOPLUS*"):
G: $U_i = [U_{i-521} + U_{i-168}] \bmod (2^{32})$

FROM: Wunderlich, J.T. (2003). Functional verification of SMP, MPP, and vector-register supercomputers through controlled randomness. In *Proceedings of IEEE SoutheastCon, Ocho Rios, Jamaica*, M. Curtis (Ed.): (pp. 117-122). IEEE Press.

and: Wunderlich, J.T. (1997). Random number generator macros for the system assurance kernel product assurance macro interface. Systems Programmer's User Manual for IBM S/390 Systems Architecture Verification, Poughkeepsie, NY.

Read more here:

Wunderlich, J.T. (2003). **Functional verification of SMP, MPP, and vector-register supercomputers through controlled randomness**. In *Proceedings of IEEE SoutheastCon*, *Ocho Rios, Jamaica*, M. Curtis (Ed.): (pp. 117-122). IEEE Press.

Wunderlich, J.T. (1997). **Random number generator macros for the system assurance kernel product assurance macro interface**. Systems Programmer's User Manual for IBM S/390 Systems Architecture Verification, Poughkeepsie, NY.