CMP 334: *Fifth Class*

Boolean formula → combinational circuit

TINY Instruction Set Architecture green card

Performance

- Metrics of performance
- Performance and execution time
- Relative performance
- CPU Time equation
- Some examples
- Averages and weighted averages
- Amdahl's law (take one)

For next class: HW 4 (begin HW 5) read A.2-, 2.1-4

Combinational Circuit Design

Combinational circuit

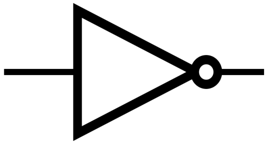
Output determined by input

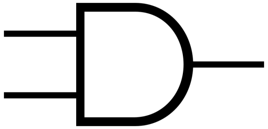
Design process

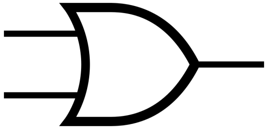
1. Specify semantics
   - Black Box input and output
   - Truth Table (input determines output)
2. Truth table → Boolean formula
3. **Minimize Boolean formula** (optional)
   - Boolean algebra
   - Karnaugh maps
4. **Boolean formula → combinational circuit**

Boolean Formula → Combinational Circuit

- Input wire for each variable
- For each sub-formula
  - Replace operand with wire (output from its sub-circuit)
  - Replace operator with gate with output wire

~ becomes 

& becomes 

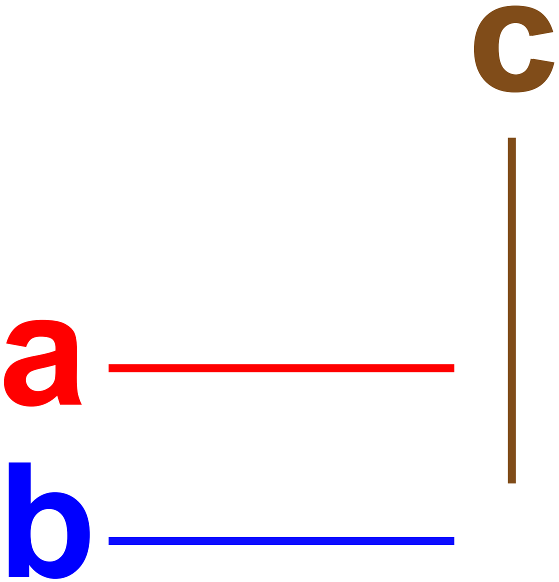
| becomes 

$$r = abc + \overline{ab}c + \overline{a}b\overline{c} + a\overline{bc}$$

$$c' = ab + ac + bc$$

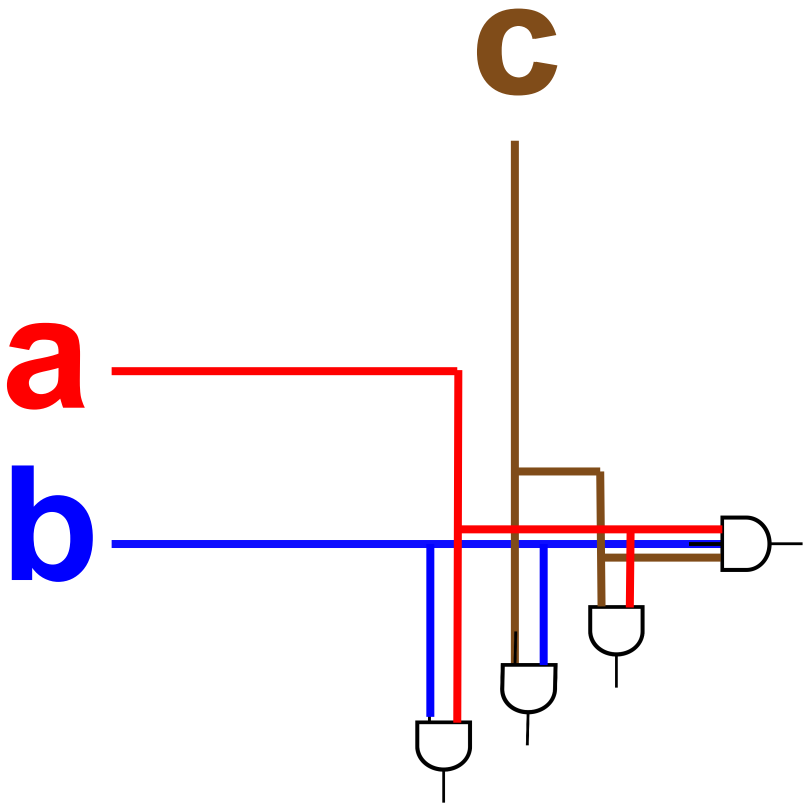
$$r = abc + \overline{ab}c + \overline{a}b\overline{c} + a\overline{bc}$$

$$c' = ab + ac + bc$$



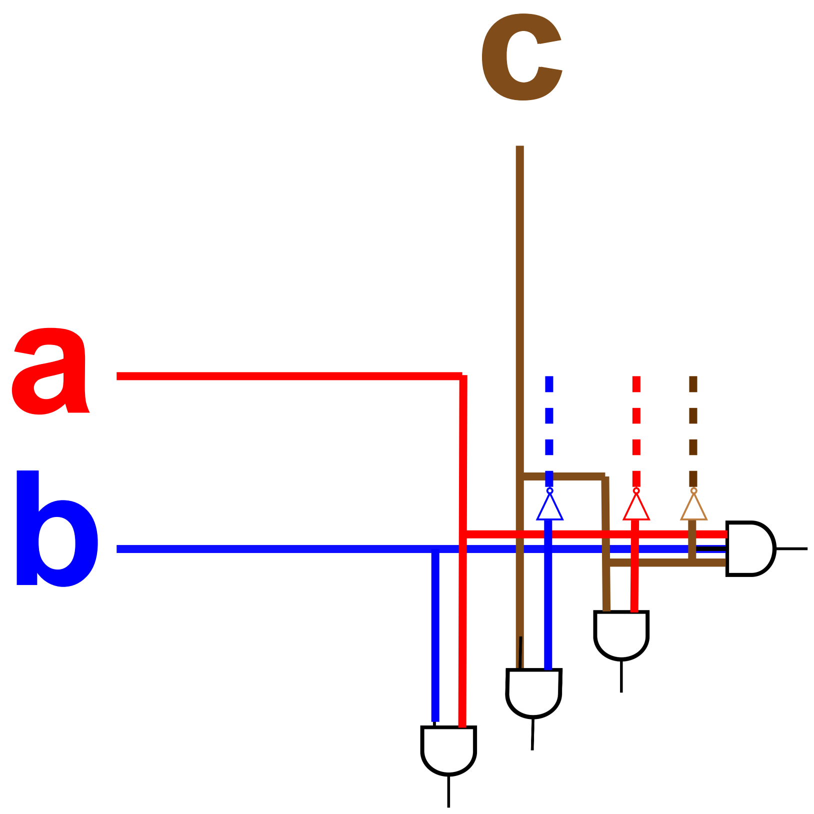
$$r = abc + \overline{ab}c + \overline{a}b\overline{c} + a\overline{bc}$$

$$c' = ab + ac + bc$$



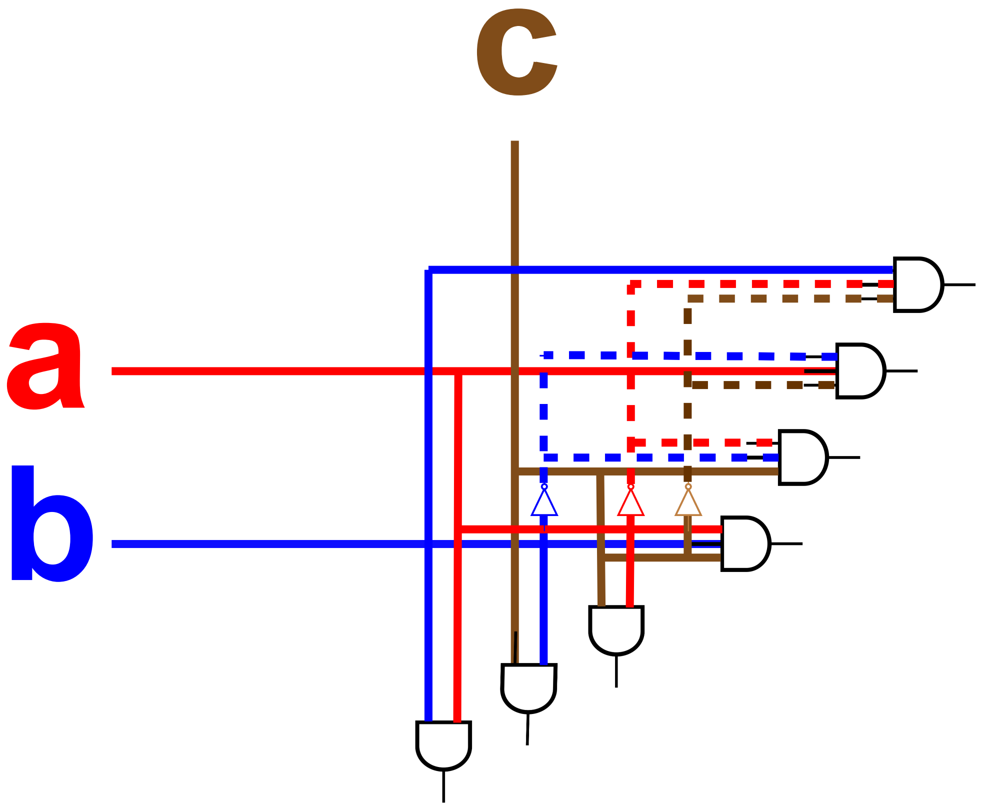
$$r = abc + \overline{ab}c + \bar{a}b\bar{c} + a\overline{bc}$$

$$c' = ab + ac + bc$$



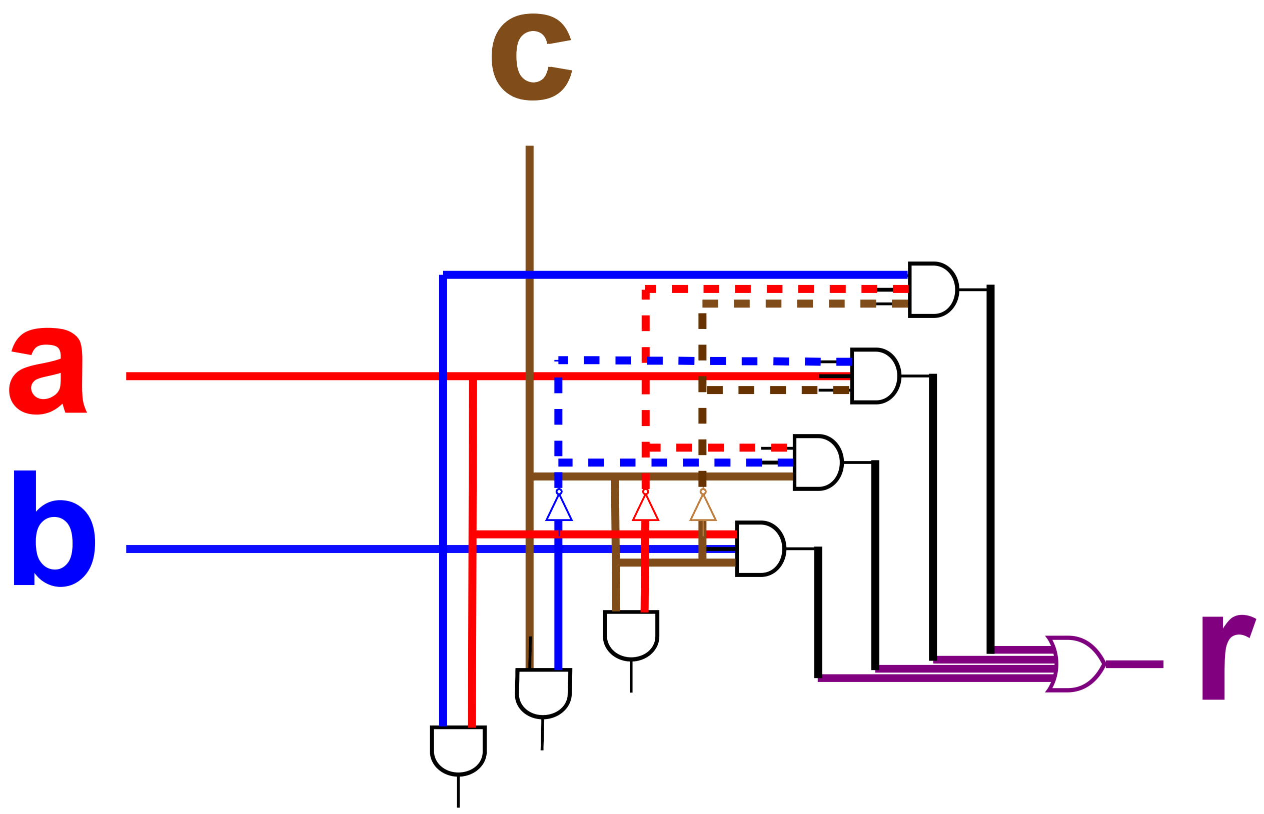
$$r = abc + \overline{ab}c + \overline{a}b\overline{c} + a\overline{bc}$$

$$c' = ab + ac + bc$$



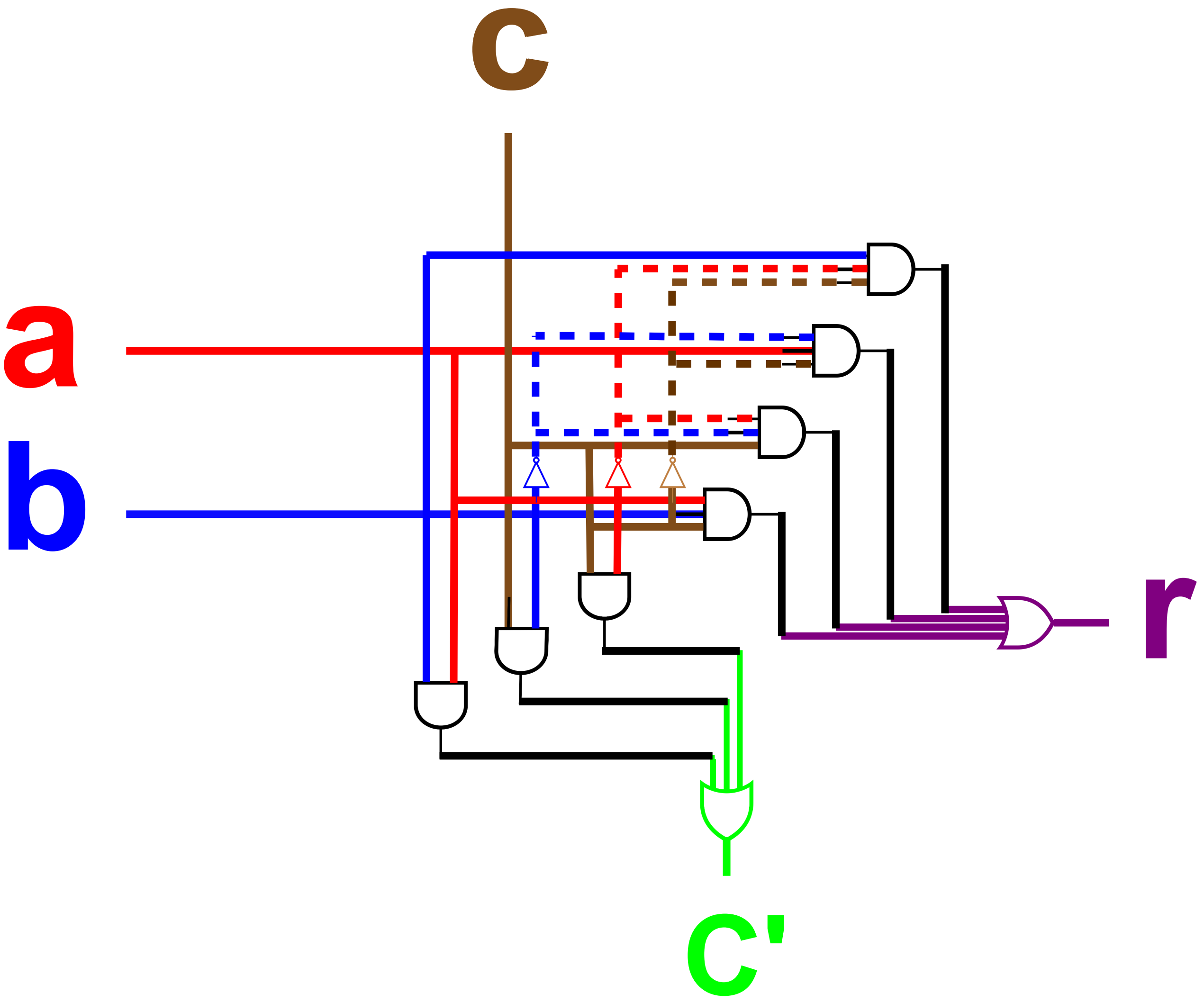
$$r = abc + \overline{ab}c + \bar{a}b\bar{c} + a\overline{bc}$$

$$c' = ab + ac + bc$$



$$r = abc + \overline{ab}c + \overline{a}b\overline{c} + a\overline{bc}$$

$$c' = ab + ac + bc$$



EGv8 Reference Data Card ("Green Card") 1. Pull along perforation to separate card 2. Fold bottom side (columns 3 and 4) together

①

LEGv8 Reference Data

CORE INSTRUCTION SET in Alphabetical Order by Mnemonic

NAME, MNEMONIC		FOR-MAT	OPCODE (9) (Hex)	OPERATION (in Verilog)	Notes
ADD	ADD	R	458	R[Rd] = R[Rn] + R[Rm]	
ADD Immediate	ADDI	I	488-489	R[Rd] = R[Rn] + ALUImm	(2,9)
ADD Immediate & Set flags	ADDIS	I	588-589	R[Rd] , FLAGS = R[Rn] + ALUImm	(1,2,9)
ADD & Set flags	ADDS	R	558	R[Rd] , FLAGS = R[Rn] + R[Rm]	(1)
AND	AND	R	450	R[Rd] = R[Rn] & R[Rm]	
AND Immediate	ANDI	I	490-491	R[Rd] = R[Rn] & ALUImm	(2,9)
AND Immediate & Set flags	ANDIS	I	790-791	R[Rd] , FLAGS = R[Rn] & ALUImm	(1,2,9)
AND & Set flags	ANDS	R	750	R[Rd] , FLAGS = R[Rn] & R[Rm]	(1)
Branch	B	B	0A0-0BF	PC = PC + BranchAddr	(3,9)
Branch conditionally	B.cond	CB	2A0-2A7	if(FLAGS==cond) PC = PC + CondBranchAddr	(4,9)
Branch with Link	BL	B	4A0-4BF	R[30] = PC + 4; PC = PC + BranchAddr	(3,9)
Branch to Register	BR	R	6B0	PC = R[Rt]	
Compare & Branch if Not Zero	CBNZ	CB	5A8-5AF	if(R[Rt]!=0) PC = PC + CondBranchAddr	(4,9)
Compare & Branch if Zero	CBZ	CB	5A0-5A7	if(R[Rt]==0) PC = PC + CondBranchAddr	(4,9)
Exclusive OR	EOR	R	650	R[Rd] = R[Rn] ^ R[Rm]	
Exclusive OR Immediate	EORI	I	690-691	R[Rd] = R[Rn] ^ ALUImm	(2,9)
LoaD Register Unscaled offset	LDUR	D	7C2	R[Rt] = M[R[Rn] + DTAddr]	(5)
LoaD Byte Unscaled offset	LDURB	D	1C2	R[Rt]={56'b0, M[R[Rn] + DTAddr](7:0)}	(5)
LoaD Half Unscaled offset	LDURH	D	3C2	R[Rt]={48'b0, M[R[Rn] + DTAddr] (15:0)}	(5)
LoaD Signed Word Unscaled offset	LDURSW	D	5C4	R[Rt] ={ 32{ M[R[Rn] + DTAddr] [31]}, M[R[Rn] + DTAddr] (31:0)}	(5)
LoaD eXclusive Register	LDXR	D	642	R[Rd] = M[R[Rn] + DTAddr]	(5,7)
Logical Shift Left	LSL	R	69B	R[Rd] = R[Rn] << shamt	
Logical Shift Right	LSR	R	69A	R[Rd] = R[Rn] >>> shamt	
MOVe wide with Keep	MOVK	IM	794-797	R[Rd] (Instruction[22:21]*16: Instruction[22:21]*16-15) = MOVImm	(6,9)
MOVe wide with Zero	MOVZ	IM	694-697	R[Rd] = { MOVImm << (Instruction[22:21]*16) }	(6,9)
Inclusive OR	ORR	R	550	R[Rd] = R[Rn] | R[Rm]	
Inclusive OR Immediate	ORRI	I	590-591	R[Rd] = R[Rn] | ALUImm	(2,9)
STore Register Unscaled offset	STUR	D	7C0	M[R[Rn] + DTAddr] = R[Rt]	(5)
STore Byte Unscaled offset	STURB	D	1C0	M[R[Rn] + DTAddr](7:0) = R[Rt](7:0)	(5)
STore Half Unscaled offset	STURH	D	3C0	M[R[Rn] + DTAddr](15:0) = R[Rt](15:0)	(5)
STore Word Unscaled offset	STURW	D	5C0	M[R[Rn] + DTAddr](31:0) = R[Rt](31:0)	(5)
STore eXclusive Register	STXR	D	640	M[R[Rn] + DTAddr] = R[Rt]; R[Rm] = (atomic) ? 0 : 1	(5,7)
SUBtract	SUB	R	658	R[Rd] = R[Rn] - R[Rm]	
SUBtract Immediate	SUBI	I	688-689	R[Rd] = R[Rn] - ALUImm	(2,9)
SUBtract Immediate & Set flags	SUBIS	I	788-789	R[Rd] , FLAGS = R[Rn] - ALUImm	(1,2,9)
SUBtract & Set flags	SUBS	R	758	R[Rd] , FLAGS = R[Rn] - R[Rm]	(1)

(1) FLAGS are 4 condition codes set by the ALU operation: Negative, Zero, oVerflow, Carry

(2) ALUImm = { 52'b0, ALU_immediate }

(3) BranchAddr = { 36{BR_address [25]}, BR_address, 2'b0 }

(4) CondBranchAddr = { 43{COND_BR_address [25]}, COND_BR_address, 2'b0 }

(5) DTAddr = { 55{DT_address [8]}, DT_address }

(6) MOVImm = { 48'b0, MOV_immediate }

(7) Atomic test&set pair; R[Rm] = 0 if pair atomic, 1 if not atomic

(8) Operands considered unsigned numbers (vs. 2's complement)

(9) Since I, B, and CB instruction formats have opcodes narrower than 11 bits, they occupy a range of 11-bit opcodes

(10) If neither is operand a NaN and Value1 == Value2, FLAGS = 4'b0110;
If neither is operand a NaN and Value1 < Value2, FLAGS = 4'b1000;
If neither is operand a NaN and Value1 > Value2, FLAGS = 4'b0010;
If an operand is a Nan, operands are unordered

ARITHMETIC CORE INSTRUCTION SET

②

NAME, MNEMONIC		FOR-MAT	OPCODE/ SHAMT (Hex)	OPERATION (in Verilog)	Notes
Floating-point ADD Single	FADDS	R	0F1 / 0A	S[Rd] = S[Rn] + S[Rm]	
Floating-point ADD Double	FADDD	R	0F3 / 0A	D[Rd] = D[Rn] + D[Rm]	
Floating-point CoMPare Single	FCMPS	R	0F1 / 08	FLAGS = (S[Rn] vs S[Rm])	(1,10)
Floating-point CoMPare Double	FCMPD	R	0F3 / 08	FLAGS = (D[Rn] vs D[Rm])	(1,10)
Floating-point DIVide Single	FDIVS	R	0F1 / 06	S[Rd] = S[Rn] / S[Rm]	
Floating-point DIVide Double	FDIVD	R	0F3 / 06	D[Rd] = D[Rn] / D[Rm]	
Floating-point MULtiply Single	FMULS	R	0F1 / 02	S[Rd] = S[Rn] * S[Rm]	
Floating-point MULtiply Double	FMULD	R	0F3 / 02	D[Rd] = D[Rn] * D[Rm]	
Floating-point SUBtract Single	FSUBS	R	0F1 / 0E	S[Rd] = S[Rn] - S[Rm]	
Floating-point SUBtract Double	FSUBD	R	0F3 / 0E	D[Rd] = D[Rn] - D[Rm]	
LoaD Single floating-point	LDURS	R	7C2	S[Rt] = M[R[Rn] + DTAddr]	(5)
LoaD Double floating-point	LDURD	R	7C0	D[Rt] = M[R[Rn] + DTAddr]	(5)
MULtiply	MUL	R	4D8 / 1F	R[Rd] = (R[Rn] * R[Rm]) (63:0)	
Signed DIVide	SDIV	R	4D6 / 02	R[Rd] = R[Rn] / R[Rm]	
Signed MULtiply High	SMULH	R	4DA	R[Rd] = (R[Rn] * R[Rm]) (127:64)	
STore Single floating-point	STURS	R	7E2	M[R[Rn] + DTAddr] = S[Rt]	(5)
STore Double floating-point	STURD	R	7E0	M[R[Rn] + DTAddr] = D[Rt]	(5)
Unsigned DIVide	UDIV	R	4D6 / 03	R[Rd] = R[Rn] / R[Rm]	(8)
Unsigned MULtiply High	UMULH	R	4DE	R[Rd] = (R[Rn] * R[Rm]) (127:64)	(8)

CORE INSTRUCTION FORMATS

R	opcode	Rm	shamt	Rn	Rd
	31 21	20 16	15 10	9 5	4 0
I	opcode	ALU_immediate		Rn	Rd
	31 22	21 10		9 5	4 0
D	opcode	DT_address	op	Rn	Rt
	31 21	20 12	11 10	9 5	4 0
B	opcode	BR_address			
	31 26	25 0			
CB	Opcode	COND_BR_address			Rt
	31 24	23 5			4 0
IW	opcode	MOV_immediate			Rd
	31 21	20 5			4 0

PSEUDOINSTRUCTION SET

NAME	MNEMONIC	OPERATION
CoMPare	CMP	FLAGS = R[Rn] - R[Rm]
CoMPare Immediate	CMPI	FLAGS = R[Rn] - ALUImm
LoaD Address	LDA	R[Rd] = R[Rn] + DTAddr
MOVe	MOV	R[Rd] = R[Rn]

REGISTER NAME, NUMBER, USE, CALL CONVENTION

NAME	NUMBER	USE	PRESERVED ACROSS A CALL?
X0 – X7	0-7	Arguments / Results	No
X8	8	Indirect result location register	No
X9 – X15	9-15	Temporaries	No
X16 (IP0)	16	May be used by linker as a scratch register; other times used as temporary register	No
X17 (IP1)	17	May be used by linker as a scratch register; other times used as temporary register	No
X18	18	Platform register for platform independent code; otherwise a temporary register	No
X19-X27	19-27	Saved	Yes
X28 (SP)	28	Stack Pointer	Yes
X29 (FP)	29	Frame Pointer	Yes
X30 (LR)	30	Return Address	Yes
XZR	31	The Constant Value 0	N.A.

OPCODES IN NUMERICAL ORDER BY OPCODE

③

Instruction Mnemonic	Format	Opcode Width (bits)	Opcode Binary	Shamt Binary	11-bit Opcode Range (1) Start (Hex)	End (Hex)
B	B	6	000101		0A0	0BF
FMULS	R	11	00011110001	000010	0F1	
FDIVS	R	11	00011110001	000110	0F1	
FCMPS	R	11	00011110001	001000	0F1	
FADDS	R	11	00011110001	001010	0F1	
FSUBS	R	11	00011110001	001110	0F1	
FMULD	R	11	00011110011	000010	0F3	
FDIVD	R	11	00011110011	000110	0F3	
FCMPD	R	11	00011110011	001000	0F3	
FADDD	R	11	00011110011	001010	0F3	
FSUBD	R	11	00011110011	001110	0F3	
STURB	D	11	00111000000		1C0	
LDURB	D	11	00111000010		1C2	
B.cond	CB	8	01010100		2A0	2A7
STURH	D	11	01111000000		3C0	
LDURH	D	11	01111000010		3C2	
AND	R	11	10001010000		450	
ADD	R	11	10001011000		458	
ADDI	I	10	1001000100		488	489
ANDI	I	10	1001001000		490	491
BL	B	6	100101		4A0	4BF
SDIV	R	11	10011010110	000010	4D6	
UDIV	R	11	10011010110	000011	4D6	
MUL	R	11	10011011000	011111	4D8	
SMULH	R	11	10011011010		4DA	
UMULH	R	11	10011011110		4DE	
ORR	R	11	10101010000		550	
ADDS	R	11	10101011000		558	
ADDIS	I	10	1011000100		588	589
ORRI	I	10	1011001000		590	591
CBZ	CB	8	10110100		5A0	5A7
CBNZ	CB	8	10110101		5A8	5AF
STURW	D	11	10111000000		5C0	
LDURSW	D	11	10111000100		5C4	
STURS	R	11	10111100000		5E0	
LDURS	R	11	10111100010		5E2	
STXR	D	11	11001000000		640	
LDXR	D	11	11001000010		642	
EOR	R	11	11001010000		650	
SUB	R	11	11001011000		658	
SUBI	I	10	1101000100		688	689
EORI	I	10	1101001000		690	691
MOVZ	IM	9	110100101		694	697
LSR	R	11	11010011010		69A	
LSL	R	11	11010011011		69B	
BR	R	11	11010110000		6B0	
ANDS	R	11	11101010000		750	
SUBS	R	11	11101011000		758	
SUBIS	I	10	1111000100		788	789
ANDIS	I	10	1111001000		790	791
MOVK	IM	9	111100101		794	797
STUR	D	11	11111000000		7C0	
LDUR	D	11	11111000010		7C2	
STURD	R	11	11111100000		7E0	
LDURD	R	11	11111100010		7E2	

(1) Since I, B, and CB instruction formats have opcodes narrower than 11 bits, they occupy a range of 11-bit opcodes, e.g., the 6-bit B format occupies 32 (2^5) 11-bit opcodes.

IEEE 754 FLOATING-POINT STANDARD

④

$(-1)^S \times (1 + \text{Fraction}) \times 2^{(\text{Exponent} - \text{Bias})}$

where Single Precision Bias = 127,
Double Precision Bias = 1023

IEEE 754 Symbols

Exponent	Fraction	Object
0	0	± 0
0	≠ 0	± Denorm
1 to MAX - 1	anything	± Fl. Pt. Num.
MAX	0	± ∞
MAX	≠ 0	NaN

S.P. MAX = 255, D.P. MAX = 2047

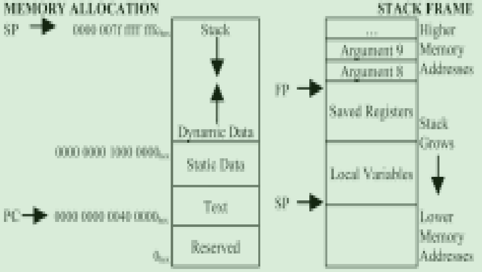
IEEE Single Precision and Double Precision Formats:

S	Exponent	Fraction
31	30 23	22 0

S	Exponent	Fraction
63	62 52	51 0

MEMORY ALLOCATION

STACK FRAME



DATA ALIGNMENT

Double Word							
Word				Word			
Halfword		Halfword		Halfword		Halfword	
Byte	Byte	Byte	Byte	Byte	Byte	Byte	Byte
0	1	2	3	4	5	6	7

Value of three least significant bits of byte address (Big Endian)

EXCEPTION SYNDROME REGISTER (ESR)

Exception Class (EC)	Instruction Length (IL)	Instruction Specific Syndrome field (ISS)
31 26	25	24 0

EXCEPTION CLASS

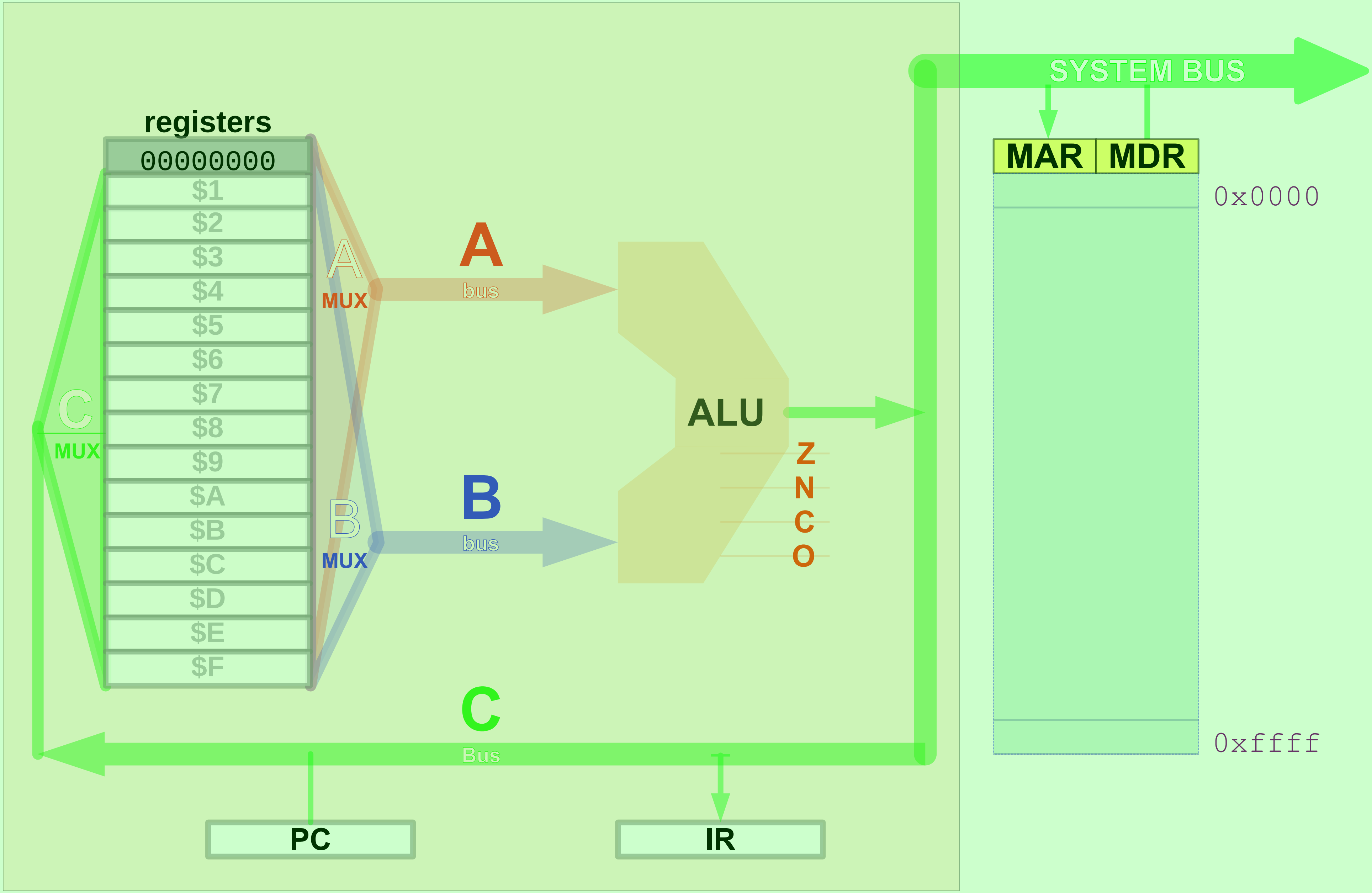
EC	Class	Cause of Exception	Number	Name	Cause of Exception
0	Unknown	Unknown	34	PC	Misaligned PC exception
7	SIMD	SIMD/FP registers disabled	36	Data	Data Abort
14	FPE	Illegal Execution State	40	FPE	Floating-point exception
17	Sys	Supervisor Call Exception	52	WPT	Data Breakpoint exception
32	Instr	Instruction Abort	56	BKPT	SW Breakpoint Exception

SIZE PREFIXES AND SYMBOLS

SIZE	PREFIX	SYMBOL	SIZE	PREFIX	SYMBOL
10^3	Kilo-	K	2^{10}	Kibi-	Ki
10^6	Mega-	M	2^{20}	Mebi-	Mi
10^9	Giga-	G	2^{30}	Gibi-	Gi
10^{12}	Tera-	T	2^{40}	Tebi-	Ti
10^{15}	Peta-	P	2^{50}	Pebi-	Pi
10^{18}	Exa-	E	2^{60}	Exbi-	Ei
10^{21}	Zetta-	Z	2^{70}	Zebi-	Zi
10^{24}	Yotta-	Y	2^{80}	Yobi-	Yi
10^{-3}	milli-	m	10^{-15}	femto-	f
10^{-6}	micro-	μ	10^{-18}	atto-	a
10^{-9}	nano-	n	10^{-21}	zepto-	z
10^{-12}	pico-	p	10^{-24}	yocto-	y

EGv8 Reference Data Card ("Green Card") 1. Pull along perforation to separate card 2. Fold bottom side (columns 3 and 4) together

The TINY Computer



TINY Instruction Set Architecture

Reference Data Card

Main Memory 65536 16-bit words

M[n] – nth memory address

^M[n] – content of M[n]

Register File 16 16-bit "registers"

15 real registers:	$1 … $F
1 pseudo-register:	$0 [$0] = 0

Immediate values

In	— n-bit signed int
Un	— n-bit unsigned int
CC	— 4-bit condition code

Instructions [0]

ADD	rT ← [rA]+[rB] [1,2]
AND	rT ← [rA]&[rB] [1,3]
BRC	PC ← [rA]+U4+1 iff CC
BRU	rL ← PC, PC ← [rA]+[rB] [1]
LDI	rT ← ^M[[rA]+U4+1] [1]
LDX	rT ← ^M[[rA]+[rB]] [1]
LIH	rT$_{15..8}$ ← I8 [1]
NOR	rT ← $\overline{\texttt{[rA]}\vert\texttt{[rB]}}$ [1,3]
SLL	rT ← [rA]<<I4 [1,3]
SRS	rT ← [rA]>>I4 [1,3]
SRU	rT ← [rA]>>>I4 [1,3]
STI	M[[rA]+U4+1] ← [rS]
STX	M[[rA]+[rB]] ← [rS]
SUB	rT ← [rA]-[rB] [1, 2]
SYS	system call [4]

Arithmetic / Logical

0100	**ADD**	rT	rA	rB
0101	**SUB**	rT	rA	rB
0110	**AND**	rT	rA	rB
0111	**NOR**	rT	rA	rB

Shift / Load Immediate

1000	**LIH**	rT	I8	
1001	**SLL**	rT	rA	U4
1010	**SRS**	rT	rA	U4
1011	**SRU**	rT	rA	U4

Load/Store

0111	**LDI**	rT	rA	U4
0110	**LDX**	rT	rA	rB
0101	**STI**	rS	rA	U4
0100	**STX**	rS	rA	RB

Branch/Special

0011	**BRC**	C	rA	U4
0010	**BCU**	rL	rA	rB
0001	reserved			
0000	**SYS**	U12		

Condition Codes

0000	**true**	TT
0001	**false**	FF
0010	A = B signed	EQ
0011	A B signed	NE
0100	A < B signed	LT
0101	A B signed	GE
0110	A B signed	LE
0111	A > B signed	GT
1000	true	
1001	false	
1010	A = B unsigned	
1011	A B unsigned	
1100	A < B unsigned	LTU
1101	A B unsigned	GEU
1110	A B unsigned	LEU
1111	A > B unsigned	GTU

Notes

[0] PC ← PC+1 ***before*** instruction execution

[1] $0 ***not*** changed

[2] Determines flags: **z, n, c, o**

[3] Determines flags: **z, n,**

[4] No op iff U12 = 0

Understanding Performance

- From *qualitative* to *quantitative* analysis
- Performance metrics (what to measure)
  - What does “performance” mean?
- Performance equations
  - ***Relative performance***
  - ***CPU time equation***
  - ***Amdahl's law***
- Statistical tools
  - **Average** and **weighted average**

Performance Metrics

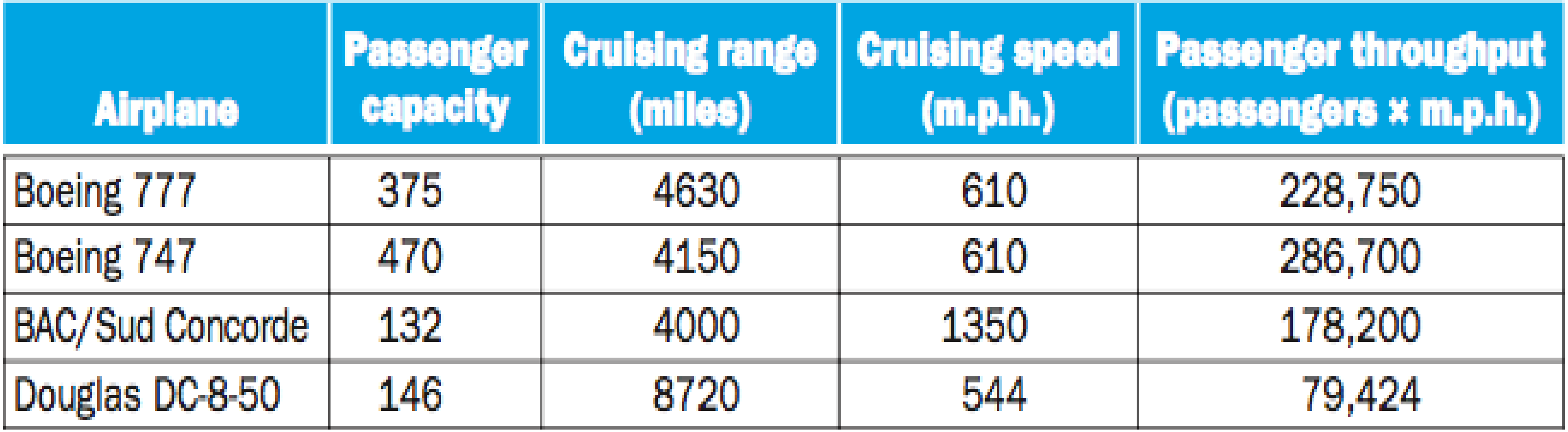
Different measures of airplane "performance"?

Speed (mph) ?

Range (miles) ?

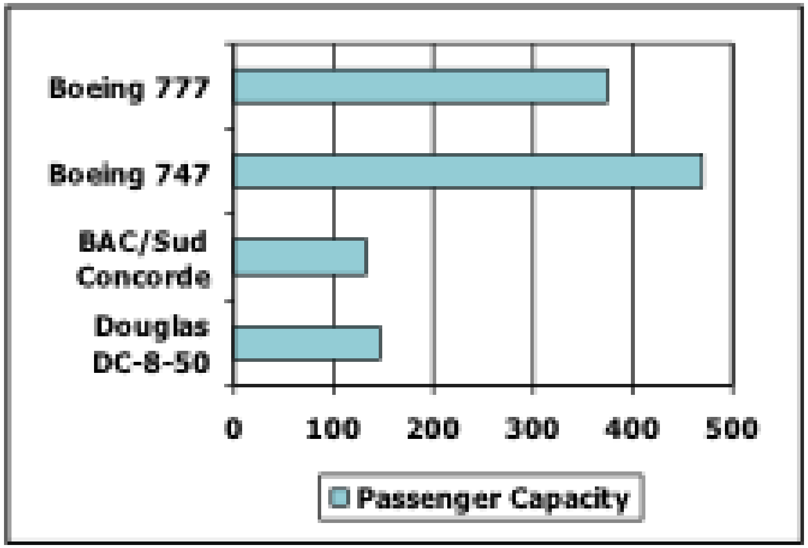
Capacity (passengers) ?

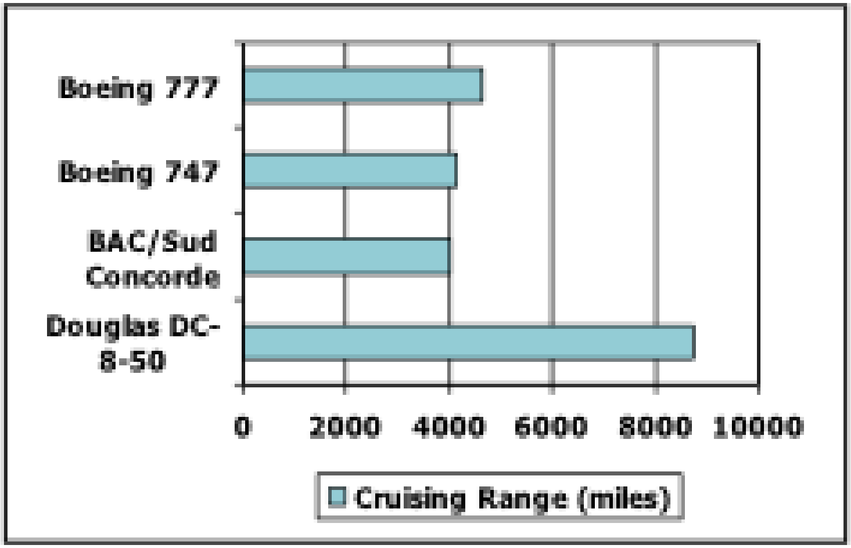
Throughput (passengers miles per hour) ?

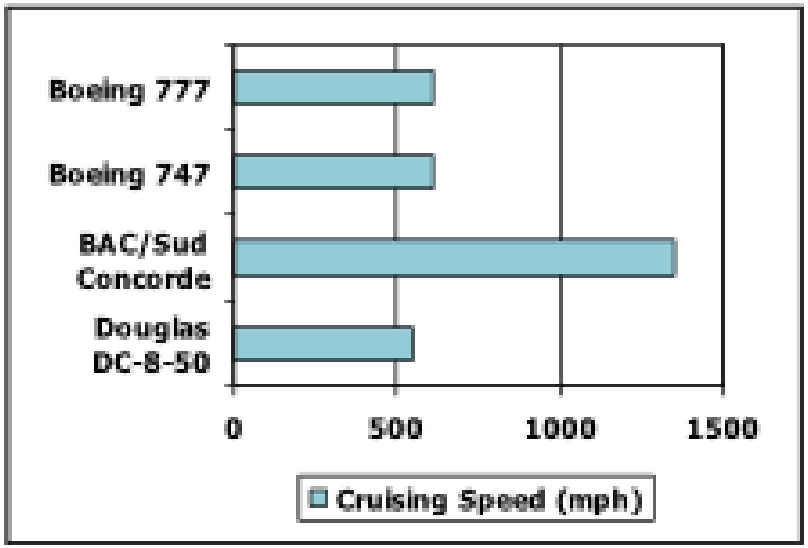


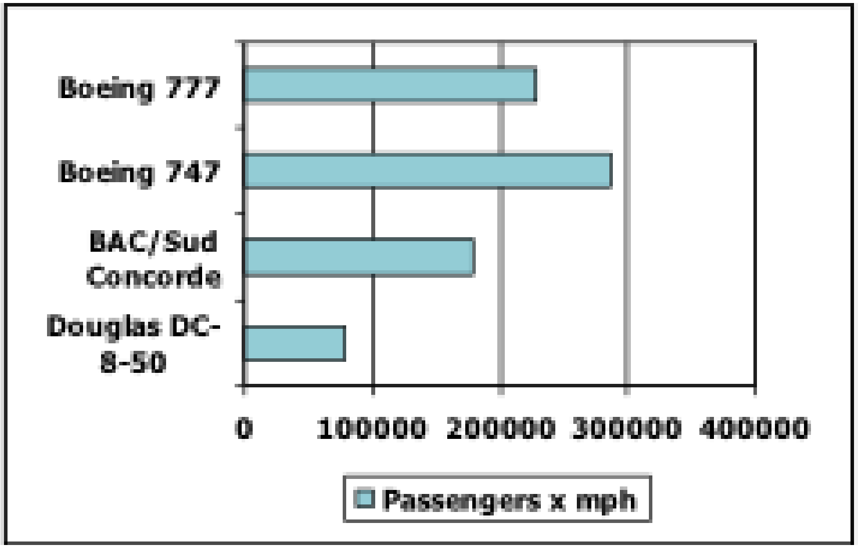
Airplane	Passenger capacity	Cruising range (miles)	Cruising speed (m.p.h.)	Passenger throughput (passengers × m.p.h.)
Boeing 777	375	4630	610	228,750
Boeing 747	470	4150	610	286,700
BAC/Sud Concorde	132	4000	1350	178,200
Douglas DC-8-50	146	8720	544	79,424

Airplane Performance Metrics









Computer Performance Metrics

Execution (response) **time** (seconds)

CPU$_{\text{time}}$ + I/O$_{\text{time}}$

Throughput (tasks per hour)

Availability (percent) $\frac{MTTF}{MTTF + MTTR}$

MTTF — Mean Time To Failure (years)

MTTR — Mean Time To Repair (minutes)

Execution energy (joules)

Throughput cost (tasks per hour per dollar)

. . .

Execution Time & Performance

Definition

$$\text{Performance}_X \equiv \frac{1}{\text{ExecutionTime}_X} \qquad \text{P}_X \equiv \frac{1}{\text{E}_X}$$

Better performance mean *shorter* execution time

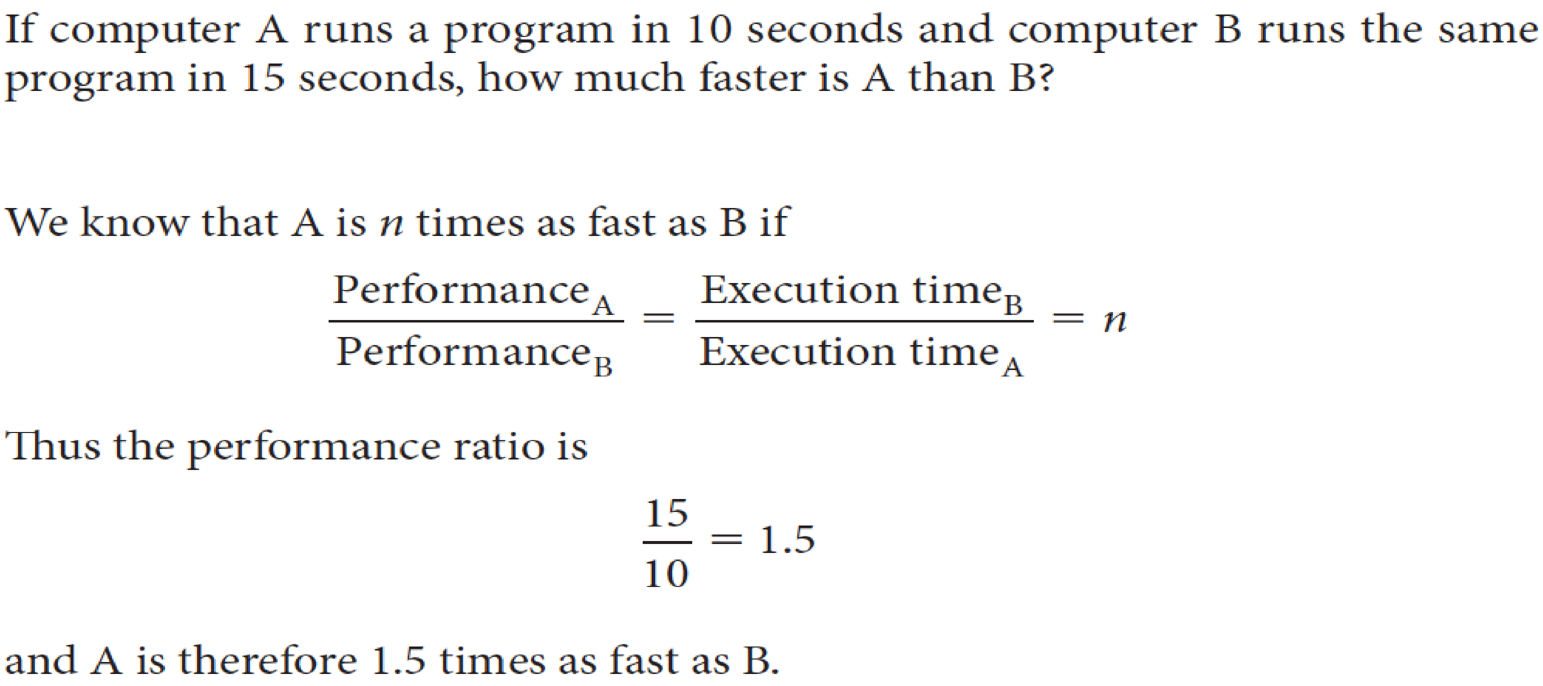
Relative performance

X is ***n*** times *as fast as* Y if and only if

$$\boldsymbol{n} = \frac{\text{P}_X}{\text{P}_Y} = \frac{\text{E}_Y}{\text{E}_X}$$

Y takes ***n*** times as long as X to execute

Relative Performance



If computer A runs a program in 10 seconds and computer B runs the same program in 15 seconds, how much faster is A than B?

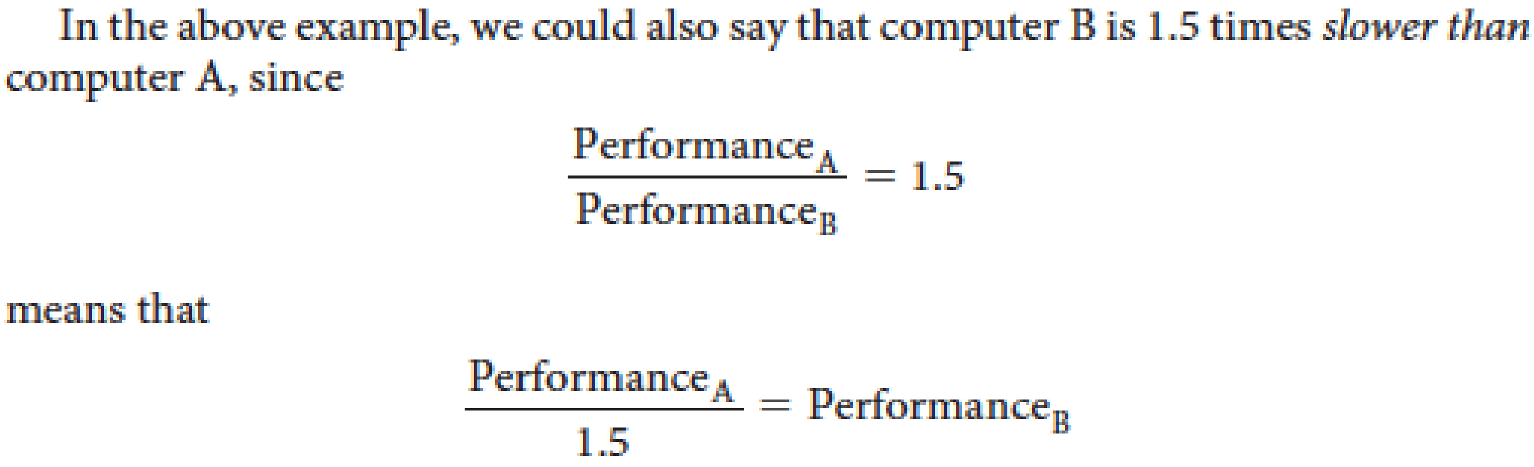
We know that A is n times as fast as B if

$$\frac{\text{Performance}_A}{\text{Performance}_B} = \frac{\text{Execution time}_B}{\text{Execution time}_A} = n$$

Thus the performance ratio is

$$\frac{15}{10} = 1.5$$

and A is therefore 1.5 times as fast as B.



In the above example, we could also say that computer B is 1.5 times *slower than* computer A, since

$$\frac{\text{Performance}_A}{\text{Performance}_B} = 1.5$$

means that

$$\frac{\text{Performance}_A}{1.5} = \text{Performance}_B$$

CPU Time Equation

Program execution time = CPU_{time} + I/0_{time}

CPU_{time} — key metric of **processor** performance

We will return to I/O_{time} later in the course

CPU_{time} = # instructions • (average) $\text{instruction}_{\text{time}}$

$\text{instruction}_{\text{time}}$ = (average) cycles per instruction • $\text{cycle}_{\text{time}}$

$$\text{cycle}_{\text{time}} = \frac{\#\ \text{seconds}}{\text{cycle}} = \frac{1}{\text{clock}_{\text{rate}}} \quad \text{(seconds)}$$

$\text{clock}_{\text{rate}}$ (**Hertz** — cycles per second)

$$\text{CPU}_{\text{time}}\left(\text{execution}\right) = \frac{\#\ \text{instructions}}{\text{execution}} \cdot \frac{\#\ \text{cycles}}{\text{instruction}} \cdot \frac{\#\ \text{seconds}}{\text{cycle}}$$

Components of performance	Units of measure
CPU execution time for a program	Seconds for the program
Instruction count	Instructions executed for the program
Clock cycles per instruction (CPI)	Average number of clock cycles per instruction
Clock cycle time	Seconds per clock cycle

Figure 1.15 shows the basic measurements at different levels in the computer and what is being measured in each case. We can see how these factors are combined to yield execution time measured in seconds per program:

$$\text{Time} = \text{Seconds/Program} = \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Clock cycles}}{\text{Instruction}} \times \frac{\text{Seconds}}{\text{Clock cycle}}$$

Always bear in mind that the only complete and reliable measure of computer performance is time. For example, changing the instruction set to lower the instruction count may lead to an organization with a slower clock cycle time or higher CPI that offsets the improvement in instruction count. Similarly, because CPI depends on type of instructions executed, the code that executes the fewest number of instructions may not be the fastest.

Performance Equations

Performance – inverse of execution time

performance: $P_x \equiv \frac{1}{T_x}$ ***relative performance:*** $\frac{P_x}{P_y} = \frac{T_y}{T_x}$

CPU time equation

$$T_{CPU}(\text{execution}) = \frac{\#\text{ instructions}}{\text{execution}} \cdot \frac{\#\text{ cycles}}{\text{instruction}} \cdot \frac{\#\text{ seconds}}{\text{cycle}}$$

Amdahl's law

$$T_{new} = \frac{\text{fraction affected} \cdot T_{old}}{\text{improvement}} + \text{fraction not affected} \cdot T_{old}$$

Relative CPU$_{\text{time}}$ Performance

$$\boxed{T_{CPU}\left(\text{execution}\right) = \frac{\#\ \text{instructions}}{\text{execution}} \cdot \frac{\#\ \text{cycles}}{\text{instruction}} \cdot \frac{\#\ \text{seconds}}{\text{cycle}}}$$

$$T_X = \#\ \text{instructions}_X \cdot \text{CPI}_X \cdot \text{cycleTime}_X$$

$$= \frac{\#\ \text{instructions}_X \cdot \text{CPI}_X}{\text{clockRate}_X}$$

$$\boxed{\frac{P_X}{P_Y} = \frac{T_Y}{T_X} = \frac{\#\ \text{instructions}_Y \cdot \text{CPI}_Y \cdot \text{cycleTime}_Y}{\#\ \text{instructions}_X \cdot \text{CPI}_X \cdot \text{cycleTime}_X}}$$

$$\boxed{\frac{P_X}{P_Y} = \frac{T_Y}{T_X} = \frac{\#\ \text{instructions}_Y \cdot \text{CPI}_Y \cdot \text{clockRate}_X}{\#\ \text{instructions}_X \cdot \text{CPI}_X \cdot \text{clockRate}_Y}}$$

A New Computer Design

Our favorite program runs in 10 seconds on computer A, which has a 2 GHz clock. We are trying to help a computer designer build a computer, B, which will run this program in 6 seconds. The designer has determined that a substantial increase in the clock rate is possible, but this increase will affect the rest of the CPU design, causing computer B to require 1.2 times as many clock cycles as computer A for this program.

What clock rate should we tell the designer to target?

A New Computer Design

Let’s first find the number of clock cycles required for the program on A:

$$\text{CPU time}_A = \frac{\text{CPU clock cycles}_A}{\text{Clock rate}_A}$$

$$10 \text{ seconds} = \frac{\text{CPU clock cycles}_A}{2 \times 10^9 \dfrac{\text{cycles}}{\text{second}}}$$

$$\text{CPU clock cycles}_A = 10 \text{ seconds} \times 2 \times 10^9 \frac{\text{cycles}}{\text{second}} = 20 \times 10^9 \text{ cycles}$$

CPU time for B can be found using this equation:

$$\text{CPU time}_B = \frac{1.2 \times \text{CPU clock cycles}_A}{\text{Clock rate}_B}$$

$$6 \text{ seconds} = \frac{1.2 \times 20 \times 10^9 \text{ cycles}}{\text{Clock rate}_B}$$

$$\text{Clock rate}_B = \frac{1.2 \times 20 \times 10^9 \text{ cycles}}{6 \text{ seconds}} = \frac{0.2 \times 20 \times 10^9 \text{ cycles}}{\text{second}} = \frac{4 \times 10^9 \text{ cycles}}{\text{second}} = 4 \text{ GHz}$$

To run the program in 6 seconds, B must have twice the clock rate of A.

A New Computer Design

$$\boxed{\frac{P_X}{P_Y} = \frac{T_Y}{T_X} = \frac{\#\,\text{instructions}_Y \cdot \text{CPI}_Y \cdot \text{clockRate}_X}{\#\,\text{instructions}_X \cdot \text{CPI}_X \cdot \text{clockRate}_Y}}$$

$$\frac{T_A}{T_B} = \frac{10}{6} = \frac{\#\,\text{instructions} \cdot \text{CPI}_A \cdot \text{clockRate}_B}{\#\,\text{instructions} \cdot \text{CPI}_B \cdot \text{clockRate}_A}$$

$$\frac{10}{6} = \frac{\cancel{\#\,\text{instructions}} \cdot \cancel{\text{CPI}_A} \cdot \text{clockRate}_B}{\cancel{\#\,\text{instructions}} \cdot 1.2 \cdot \cancel{\text{CPI}_A} \cdot 2\ GHz}$$

$$\text{clockRate}_B = \frac{10 \cdot 1.2 \cdot 2}{6}\ \text{GHz} = 4\ \text{GHz}$$

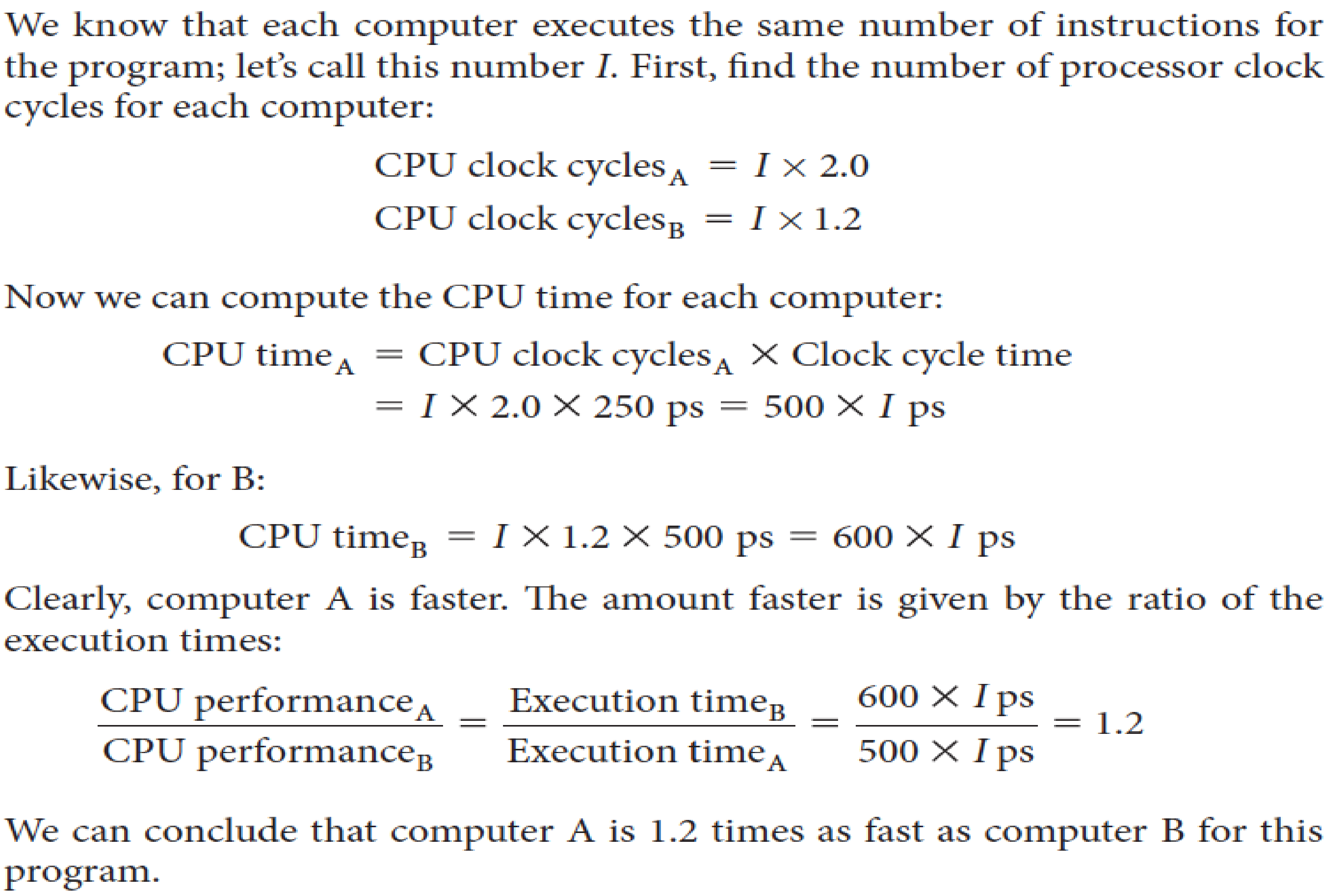
Which Computer is Faster

Suppose we have two implementations of the same instruction set architecture.

Computer A has a clock cycle time of 250 ps and a CPI of 2.0 for some program, and computer B has a clock cycle time of 500 ps and a CPI of 1.2 for the same program.

Which computer is faster for this program and by how much?

Which Computer is Faster?



We know that each computer executes the same number of instructions for the program; let's call this number I. First, find the number of processor clock cycles for each computer:

$$\text{CPU clock cycles}_A = I \times 2.0$$
$$\text{CPU clock cycles}_B = I \times 1.2$$

Now we can compute the CPU time for each computer:

$$\begin{aligned}\text{CPU time}_A &= \text{CPU clock cycles}_A \times \text{Clock cycle time} \\ &= I \times 2.0 \times 250 \text{ ps} = 500 \times I \text{ ps}\end{aligned}$$

Likewise, for B:

$$\text{CPU time}_B = I \times 1.2 \times 500 \text{ ps} = 600 \times I \text{ ps}$$

Clearly, computer A is faster. The amount faster is given by the ratio of the execution times:

$$\frac{\text{CPU performance}_A}{\text{CPU performance}_B} = \frac{\text{Execution time}_B}{\text{Execution time}_A} = \frac{600 \times I \text{ ps}}{500 \times I \text{ ps}} = 1.2$$

We can conclude that computer A is 1.2 times as fast as computer B for this program.

Which Computer is Faster?

$$\boxed{\frac{P_X}{P_Y} = \frac{T_Y}{T_X} = \frac{\#\,\text{instructions}_Y \cdot \text{CPI}_Y \cdot \text{cycleTime}_Y}{\#\,\text{instructions}_X \cdot \text{CPI}_X \cdot \text{cycleTime}_X}}$$

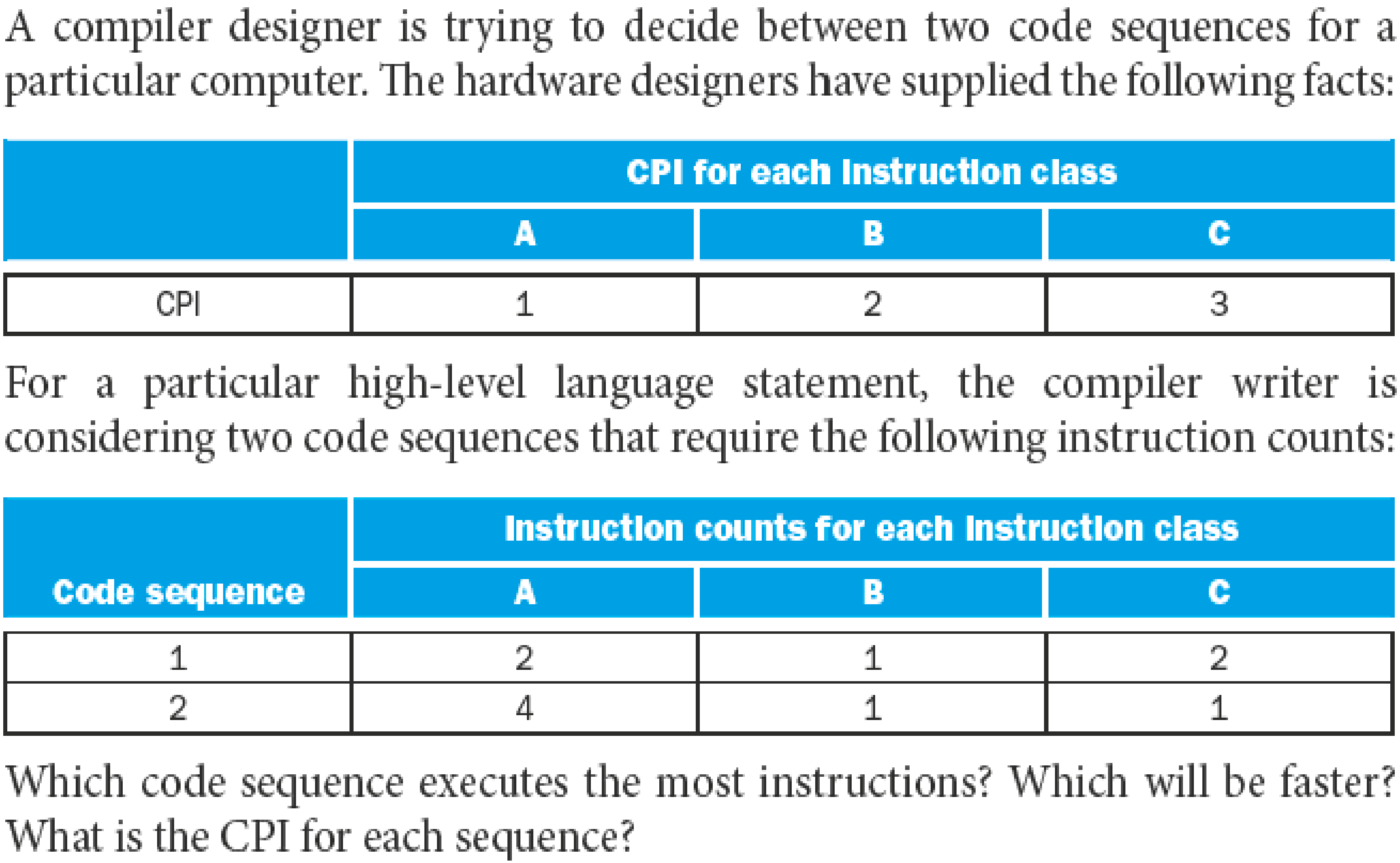
$$\frac{P_A}{P_B} = \frac{\#\,\text{instructions} \cdot \text{CPI}_B \cdot \text{cycleTime}_B}{\#\,\text{instructions} \cdot \text{CPI}_A \cdot \text{cycleTime}_A}$$

$$= \frac{\cancel{\#\,\text{instructions}} \cdot 1.2 \cdot 500\ \cancel{\text{ps}}}{\cancel{\#\,\text{instructions}} \cdot 2.0 \cdot 250\ \cancel{\text{ps}}}$$

$$= \frac{1.2 \cdot 500}{2.0 \cdot 250} = \frac{600}{500} = 1.2$$

Computer A is 1.2 times faster that Computer B

Comparing Code Segments



A compiler designer is trying to decide between two code sequences for a particular computer. The hardware designers have supplied the following facts:

	CPI for each instruction class		
	A	B	C
CPI	1	2	3

For a particular high-level language statement, the compiler writer is considering two code sequences that require the following instruction counts:

	Instruction counts for each instruction class		
Code sequence	A	B	C
1	2	1	2
2	4	1	1

Which code sequence executes the most instructions? Which will be faster? What is the CPI for each sequence?

Sequence 1 executes 2 + 1 + 2 = 5 instructions. Sequence 2 executes 4 + 1 + 1 = 6 instructions. Therefore, sequence 1 executes fewer instructions.

We can use the equation for CPU clock cycles based on instruction count and CPI to find the total number of clock cycles for each sequence:

$$\text{CPU clock cycles} = \sum_{i=1}^{n} (\text{CPI}_i \times \text{C}_i)$$

This yields

$$\text{CPU clock cycles}_1 = (2 \times 1) + (1 \times 2) + (2 \times 3) = 2 + 2 + 6 = 10 \text{ cycles}$$

$$\text{CPU clock cycles}_2 = (4 \times 1) + (1 \times 2) + (1 \times 3) = 4 + 2 + 3 = 9 \text{ cycles}$$

So code sequence 2 is faster, even though it executes one extra instruction. Since code sequence 2 takes fewer overall clock cycles but has more instructions, it must have a lower CPI. The CPI values can be computed by

$$\text{CPI} = \frac{\text{CPU clock cycles}}{\text{Instruction count}}$$

$$\text{CPI}_1 = \frac{\text{CPU clock cycles}_1}{\text{Instruction count}_1} = \frac{10}{5} = 2.0$$

$$\text{CPI}_2 = \frac{\text{CPU clock cycles}_2}{\text{Instruction count}_2} = \frac{9}{6} = 1.5$$

Comparing Code Segments

$$\boxed{T_X = \#\,\text{instructions}_X \cdot \text{CPI}_X \cdot \text{cycleTime}_X}$$

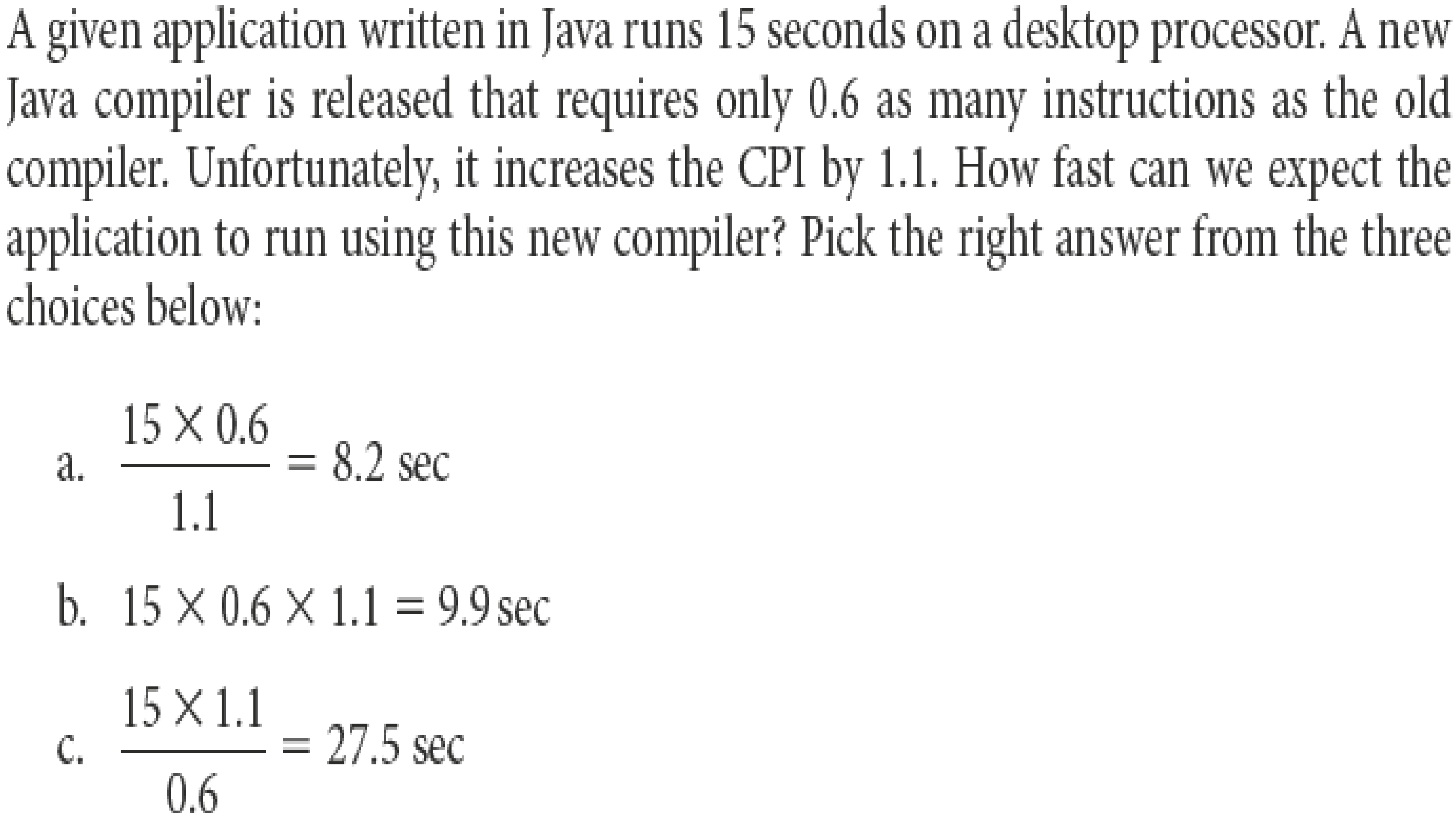
$$\begin{aligned}\text{cycles}_X &= \#\,\text{instructions}_X \cdot \text{CPI}_X \\ &= \text{A-cycles}_X + \text{B-cycles}_X + \text{C-cycles}_X\end{aligned}$$

$$\begin{aligned}\text{cycles}_1 &= \#\text{A-instr}_1 \cdot \text{CPI}_A + \#\text{B-instr}_1 \cdot \text{CPI}_B + \#\text{C-instr}_1 \cdot \text{CPI}_C \\ \text{cycles}_2 &= \#\text{A-instr}_2 \cdot \text{CPI}_A + \#\text{B-instr}_2 \cdot \text{CPI}_B + \#\text{C-instr}_2 \cdot \text{CPI}_C\end{aligned}$$

$$\text{cycles}_1 = 2 \cdot 1 + 1 \cdot 2 + 2 \cdot 3 = 10 \qquad \text{CPI}_1 = \frac{10}{5} = 2.0$$

$$\text{cycles}_2 = 4 \cdot 1 + 1 \cdot 2 + 1 \cdot 3 = 9 \qquad \text{CPI}_2 = \frac{9}{6} = 1.5$$

Check Yourself



A given application written in Java runs 15 seconds on a desktop processor. A new Java compiler is released that requires only 0.6 as many instructions as the old compiler. Unfortunately, it increases the CPI by 1.1. How fast can we expect the application to run using this new compiler? Pick the right answer from the three choices below:

a. $\dfrac{15 \times 0.6}{1.1} = 8.2 \text{ sec}$

b. $15 \times 0.6 \times 1.1 = 9.9 \text{ sec}$

c. $\dfrac{15 \times 1.1}{0.6} = 27.5 \text{ sec}$

CPU Time Equation

$$\boxed{\frac{P_X}{P_Y} = \frac{T_Y}{T_X} = \frac{\#\,\text{instructions}_Y \cdot \text{CPI}_Y \cdot \text{clock rate}_X}{\#\,\text{instructions}_X \cdot \text{CPI}_X \cdot \text{clock rate}_Y}}$$

$$\frac{T_J}{T_K} = \frac{\#\,\text{instructions}_J \cdot \text{CPI}_J \cdot \text{clock rate}_K}{\#\,\text{instructions}_K \cdot \text{CPI}_K \cdot \text{clock rate}_J}$$

$$\frac{15 \;\; \text{seconds}}{T_K} = \frac{\cancel{\#\,\text{instructions}_J} \qquad \cancel{\text{CPI}_J} \cdot \qquad \cancel{\text{clock rate}_J}}{\cancel{\#\,\text{instructions}_J} \cdot 0.6 \cdot \cancel{\text{CPI}_J} \cdot 1.1 \cdot \cancel{\text{clock rate}_J}}$$

$$T_K = 15 \cdot 0.6 \cdot 1.1 = 9.9 \;\; \text{seconds}$$

Basic Statistical Tools

Given values: $\{v_1,\ v_2,\ \ldots\ \ v_N\}$ & weights: $\{w_1,\ w_2,\ \ldots\ \ w_N\}$

average: $\vec{v}\ \equiv\ \dfrac{\sum_{i=1}^{N} v_i}{N}$

total weight: $W \equiv \sum_{i=1}^{N} w_i$ ***normalized weight:*** $q_i \equiv \dfrac{w_i}{W}$ $\left(\sum_{i=0}^{N} q_i = 1\right)$

weighted average: $\dfrac{\sum_{i=1}^{N} w_i v_i}{\sum_{i=1}^{N} w_i} = \dfrac{\sum_{i=1}^{N} w_i v_i}{W} = \sum_{i=1}^{N} \dfrac{w_i}{W} v_i = \sum_{i=1}^{N} q_i v_i$

Grade Point Average

$$GPA \equiv \frac{\sum\limits_{c \in Courses} GradePoint\left(c\right) \cdot Hours\left(c\right)}{\sum\limits_{c \in Courses} Hours\left(c\right)}$$

Typical Instruction Statistics

Instruction types, frequencies, and execution times

50% ALU instructions	5 CPI
30% Memory instructions	
20% Load	8 CPI
10% Store	6 CPI
20% Branch instructions	10 CPI
0.5% Special instructions	

Average Cycles Per Instruction

(Weighted) average CPI

$$\begin{aligned}
&= q_{ALU}T_{ALU} + q_{Load}T_{Load} + q_{Store}T_{Store} + q_{Branch}T_{Branch} \\
&= 0.5\bullet5 + 0.2\bullet8 + 0.1\bullet6 + 0.2\bullet10 \\
&= 2.5 + 1.6 + 0.6 + 2.0 \\
&= 6.7 \text{ cycles} \quad \text{approximation: } 20 / 6.7 \approx 3
\end{aligned}$$

Execution time fraction by instruction type

ALU	2.5 / 6.7	~ 37.5%
Load	1.6 / 6.7	~ 24.0%
Store	0.6 / 6.7	~ 9.0%
Branch	2.0 / 6.7	~ 30.0%

Performance Equations

Performance – inverse of execution time

performance: $P_x \equiv \frac{1}{T_x}$ ***relative performance:*** $\frac{P_x}{P_y} = \frac{T_y}{T_x}$

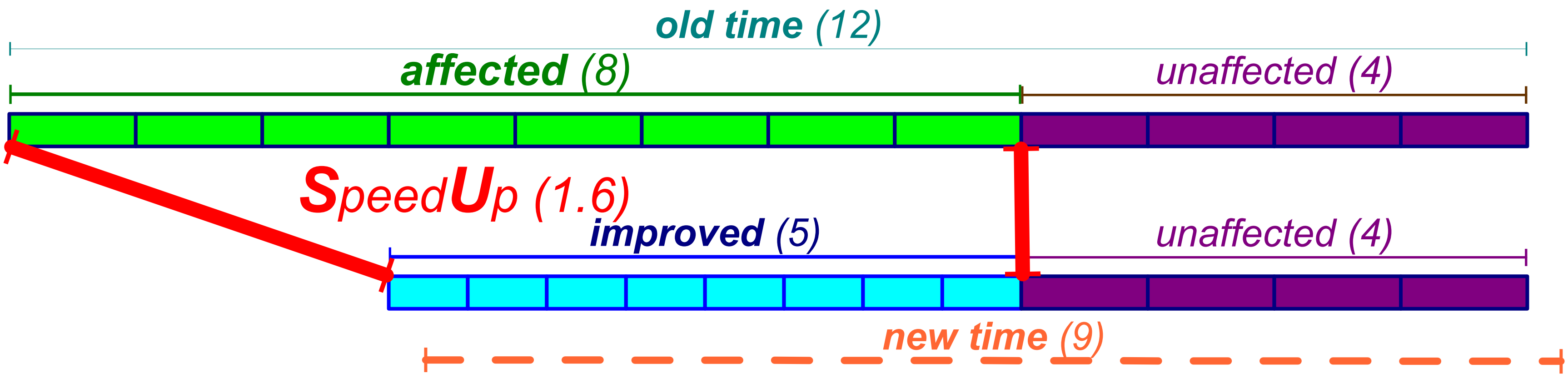
CPU time equation

$$T_{CPU}(\text{execution}) = \frac{\#\text{ instructions}}{\text{execution}} \cdot \frac{\#\text{ cycles}}{\text{instruction}} \cdot \frac{\#\text{ seconds}}{\text{cycle}}$$

Amdahl's law

$$T_{new} = \frac{\text{fraction affected} \cdot T_{old}}{\text{improvement}} + \text{fraction not affected} \cdot T_{old}$$

Amdahl's Law



old time *(12)*

affected *(8)*

unaffected (4)

SpeedUp *(1.6)*

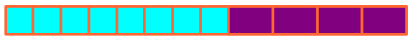
improved *(5)*

unaffected (4)

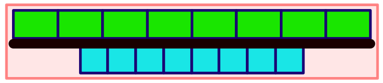
new time *(9)*

T_{old} = affected + unaffected

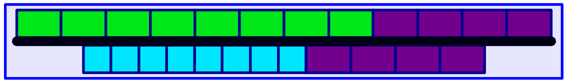
T_{new} = improved + unaffected



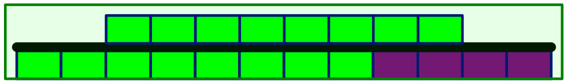
SpeedUp = affected / improved



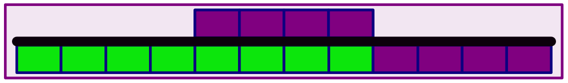
Overall SpeedUp = $T_{\text{old}} / T_{\text{new}}$



(fraction affected) F_a = affected / T_{old}



(fraction unaffected) $\bar{F}_a$ = unaffected / T_{old}



Improving a Race Car

	% time	% fuel useage	% tire ware	% miles
acceleration	5	30	10	10
cruise	90	50	50	20
brake	5	10	40	40
turns	15	10	10	30

Average Cycles Per Instruction

(Weighted) average CPI

$$= q_{ALU}T_{ALU} + q_{Load}T_{Load} + q_{Store}T_{Store} + q_{Branch}T_{Branch}$$

$$= 0.5 \bullet 5 + 0.2 \bullet 8 + 0.1 \bullet 6 + 0.2 \bullet 10$$

$$= 2.5 + 1.6 + 0.6 + 2.0$$

$= 6.7$ cycles approximation: $20 / 6.7 \approx 3$

Execution time fraction by instruction type

ALU	2.5 / 6.7	~ 37.5%
Load	1.6 / 6.7	~ 24.0%
Store	0.6 / 6.7	~ 9.0%
Branch	2.0 / 6.7	~ 30.0%

CPU Time Equation

$$\boxed{T_{CPU}(\text{execution}) = \frac{\#\,\text{instructions}}{\text{execution}} \cdot \frac{\#\,\text{cycles}}{\text{instruction}} \cdot \frac{\#\,\text{seconds}}{\text{cycle}}}$$

$$\text{If } T_{CPU}(\text{execution}) \approx 20 \text{ seconds}, \ \ \text{cycle}_{\text{time}} = 10^{-9} \text{ seconds}$$

$$20 \ \ \text{seconds} \ \approx \ \#\,\text{instructions} \cdot 6.7 \cdot 10^{-9} \ \ \text{seconds}$$

$$\#\,\text{instructions} \ \approx \ \frac{20}{6.7 \cdot 10^{-9}} \ \approx \ 3 \cdot 10^{9}$$

$$\boxed{\text{instruction}_{\text{time}} = \frac{\#\,\text{seconds}}{\text{instruction}} = \frac{\#\,\text{cycles}}{\text{instruction}} \cdot \frac{\#\,\text{seconds}}{\text{cycle}}}$$

Amdahl's Law 1

$$\boxed{T_{new} = \frac{\text{fraction affected} \cdot T_{old}}{\text{improvement}} + \text{fraction not affected} \cdot T_{old}}$$

Improvement X

reduces ALU instructions time from 5 to 4 ns

$$T_X = \frac{\text{fraction affected} \cdot 20\ sec}{\text{improvement}} + \text{fraction not affected} \cdot 20\ sec$$

$$T_X = \left(\frac{\frac{2.5}{6.7}\,20}{\frac{5}{4}} + \frac{4.2}{6.7}\,20 \right) sec \approx \left(\frac{7.5}{1.25} + 12.6 \right) sec = 18.6\ sec$$

Amdahl's Law 2

$$\boxed{T_{new} = \frac{\text{fraction affected} \cdot T_{old}}{\text{improvement}} + \text{fraction not affected} \cdot T_{old}}$$

Improvement Y

reduces Load instructions time from 8 to 4 ns

$$T_Y = \frac{\text{fraction affected} \cdot 20\ sec}{\text{improvement}} + \text{fraction not affected} \cdot 20\ sec$$

$$T_Y = \left(\frac{\frac{1.6}{6.7}\ 20}{\frac{8}{4}} + \frac{5.1}{6.7}\ 20 \right) sec \approx \left(\frac{4.8}{2} + 15.3 \right) sec = 17.7\ sec$$

Amdahl's Law 3

$$\boxed{T_{new} = \frac{\text{fraction affected} \cdot T_{old}}{\text{improvement}} + \text{fraction not affected} \cdot T_{old}}$$

Improvement Z

reduces Store instructions time from 6 to 2 ns

$$T_Z = \frac{\text{fraction affected} \cdot 20\ sec}{\text{improvement}} + \text{fraction not affected} \cdot 20\ sec$$

$$T_Z = \left(\frac{\frac{0.6}{6.7}\ 20}{\frac{6}{2}} + \frac{6.1}{6.7}\ 20 \right) sec \approx \left(\frac{1.8}{3} + 18.3 \right) sec = 18.9\ sec$$

Amdahl's Law 4

$$\boxed{T_{new} = \frac{\text{fraction affected} \cdot T_{old}}{\text{improvement}} + \text{fraction not affected} \cdot T_{old}}$$

Improvement W

reduces Branch instruction time from 10 to $5\ ns$

$$T_W = \frac{\text{fraction affected} \cdot 20\ sec}{\text{improvement}} + \text{fraction not affected} \cdot 20\ sec$$

$$T_W = \left(\frac{\frac{2.0}{6.7}\,20}{\frac{10}{5}} + \frac{4.7}{6.7}\,20 \right) sec \approx \left(\frac{6}{2} + 14.1 \right) sec = 17.1\ sec$$

Relative Performance

performance: $P_x \equiv \frac{1}{T_x}$ ***relative performance:*** $\frac{P_x}{P_y} = \frac{T_y}{T_x}$

$$\frac{P_X}{P_{old}} = \frac{T_{old}}{T_X} = \frac{20}{18.6} \approx 1.075$$

$$\frac{P_Y}{P_{old}} = \frac{T_{old}}{T_Y} = \frac{20}{17.7} \approx 1.130$$

$$\frac{P_Z}{P_{old}} = \frac{T_{old}}{T_Z} = \frac{20}{18.9} \approx 1.058$$

$$\frac{P_W}{P_{old}} = \frac{T_{old}}{T_Z} = \frac{20}{17.1} \approx 1.170$$