	Store Registers								Temporary Registers							
Assy Lang	$s0	$s1	$s2	$s3	$s4	$s5	$s6	$s7	$t0	$t1	$t2	$t3	$t4	$t5	$t6	$t7
Mach Lang	16	17	18	19	20	21	22	23	8	9	10	11	12	13	14	15
Decimal	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Hex	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
Binary	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111

add $rd, $rs, $rt — *add $s0, $s7, $t5* — *$s0 = 16, $s7 = 23, $t5 = 13*

op	rs	rt	rd	shamt	funct
0	23	13	16	0	32
000000	10111	01101	10000	00000	100000

0	2	14	13	8	0	2	0
0	2	E	D	8	0	2	0

sub $rd, $rs, $rt — *sub $s1, $t7, $t0* — *$s1 = 17, $t7 = 15, $t0 = 8*

op	rs	rt	rd	shamt	funct
0	15	8	17	0	34
000000	01111	01000	10001	00000	100010

0	2	14	8	8	8	2	2
0	2	E	8	8	8	2	2

sll $rd, $rt, shamt — *sll $s3, $t3, 3* — *$s3 = 19, $t3 = 11, shamt = 2^3*

op	rs	rt	rd	shamt	funct
0	0	11	19	0	0
000000	00000	01011	10011	00011	000000

0	0	0	11	9	8	12	0
0	0	0	B	9	8	C	0

addi $rt, $rs, const — *addi $s4, $t4, 128* — *$s4 = 20, $t4 = 12, const = 128*

op	rs	rt	constant / address offset
8	12	20	128
001000	01100	10100	0000000010000000

2	1	9	4	0	0	8	0
2	1	9	4	0	0	8	0

lw $rt, offset($rs) — *lw $t1, 100($s3)* — *$t1 = 9, $s3 = 19, offset = 100*

op	rs	rt	constant / address offset
35	19	9	100
100011	10011	01001	0000000001100100

8	13	6	9	0	0	6	4
8	D	6	9	0	0	6	4

sw $rt, offset($rs) — *sw $t1, 40($s3)* — *$t1 = 9, $s3 = 19, offset = 40*

op	rs	rt	constant / address offset
43	19	9	40
101011	10011	01001	0000000000101000

10	13	6	9	0	0	2	8
A	D	6	9	0	0	2	8

↑ Branch ⇒ op = 4 (same as lw)   Jump ⇒ op = 2 (6.26 bits)

X=2 and *Y=3*. If X and Y are represented by *signed 4-bit* integers, calculate X-Y using **saturating** arithmetic. Express result in hexadecimal format.

0	0	1	0	→	0	0	1	0
0	0	1	1	→	1	1	0	1
=0xF					1	1	1	1

Show the best way to calculate *(0x72*0xCC)* using shifts and adds.

0x72 = 0111 0010 → $2^6+2^5+2^4+2^1 \Rightarrow 6+5+4+1 = 16 \Rightarrow\Rightarrow$ 16 shifts and 3 adds

0xCC = 1100 1100 → $2^7+2^6+2^3+2^2 \Rightarrow 7+6+3+2 = 18 \Rightarrow\Rightarrow$ 18 shifts and 3 adds

$Best\ Way \Rightarrow (2^6+2^5+2^4+2^1)*0xCC \Rightarrow\Rightarrow (2^6*oxCC)+(2^5*0xCC)+(2^4*0xCC)+(2^1*0xCC)$

Operation	A	B	Overflow when result is	IEEE Floating Point	bias	sign	exponent	fraction
A+B	≥ 0	≥ 0	< 0	Single Precision 32 bits	127	1	8	23
A+B	< 0	< 0	> 0	Double Precision 64 bits	1023	1	11	52
A-B	≥ 0	< 0	< 0	Single min-max	-126	127		
A-B	< 0	≥ 0	≥ 0	Double min-max	-1022	1023		

Exponent Field	Fraction Field	Represents	Denormalized number
All 0s	All 0s	0	$\pm 0.fraction_2 * 2^{(exp-bias)}$
All 0s	Not All 0s	$\pm 0.fraction_2 * 2^{(1-bias)}$	Normalized number
Not all 0s & not all 1s	Any	$\pm 1.fraction_2 * 2^{(exp-bias)}$	$\pm 1.fraction_2 * 2^{(exp-bias)}$
All 1s	All 0s	infinity	
All 1s	Not All 0s	NaN	

Convert the number, 216.875 to IEEE double-precision floating point.

	0.5_{10}	2^{-1}	0.1_2	216	1	1	0	1	1	0	0	0	.	0	0	0
0.875	0.25_{10}	2^{-2}	0.01_2	216.875	1	1	0	1	1	0	0	0	.	1	1	1
	0.125_{10}	2^{-3}	0.001_2	1 .	1	0	1	1	0	0	0	1	1	1	*	2^7
	sum →		0.111_2													

$Bias = 1023 \Rightarrow ExponentField - Bias = 7 \Rightarrow ExponentField = 1023 + 7 = 1030 \Rightarrow to\ hex = 0x406$

1030/16=64.375	.375*16=6	0x406=	1	0	0	0	0	0	0	0	1	1	0
64/16=4.00	.00*16=0												
4/16=0.25	0.25*16=4												

sign	exponent = 11 bits	fraction = 52 bits
0	1 0 0 0 0 0 0 0 1 1 0	1 0 1 1 0 0 0 1 1 1 0...

$32*16 \Rightarrow 32 = 1.0_2 * 2^5 \Rightarrow 16 = 1.0_2 * 2^4 \Rightarrow (1.0_2 * 2^5)*(1.0_2 * 2^4) = 1.0_2 * 2^9$

$Bias = 1023 \Rightarrow ExponentField - Bias = 9 \Rightarrow ExponentField = 1023 + 9 = 1032 \Rightarrow to\ hex = 0x408$

1032/16=64.5	.5*16=8	0x408=	1	0	0	0	0	0	0	1	0	0	0
64/16=4.00	.00*16=0												
4/16=0.25	0.25*16=4												

sign	exponent = 11 bits	fraction = 52 bits
0	1 0 0 0 0 0 0 1 0 0 0	0 0 0 0 0 0 0 0 0 0 0...

Convert the MIPS assembly code below to C. Assume that *a, b, c, i,* and *j,* are assigned to registers *$s0, $s1, $s2, $s3,* and *$s4*, respectively. Assume that the base addresses of arrays M and K are assigned to registers *$s6* and *$s7*, respectively.

a	b	c	i	j	M[0]	K[0]
$s0	$s1	$s2	$s3	$s4	$s6	$s7

sll $t0, $s3, 2	# $t0 = i x 4
add $t1, $s7, $t0	# $t1 = &K[0] + 4i
lw $t0, 12($t1)	# $t0 = K[i + 3]
sub $t1, $s1, $t0	# t1 = b – K[i + 3]
add $s2, $s0, $t1	# c = a + b – K[i + 3]

For the following C statement, write a minimal sequence of MIPS assembly instructions that does the identical operation. Assume *$t1 = A, $t2 = B,* and *$s1* is the base address of *C*.
A = C[0] << 4;

A	B	C[0]
$t1	$t2	$s1

A = C[0];	lw $t1, 0($s1)
A = C[0]<< 4;	sll $t1, $t1, 4

Given the C statement: *g = h + Z[j]*
What is the corresponding MIPS assembly language code? Assume the variables *g, h,* and *j* are assigned to registers *$s1, $s2,* and *$s3. Z* is an m-word array, and the starting or base address of array *Z is zero*.

g	h	j	Z[0]
$s1	$s2	$s3	

sll $t0, $s3, 2	# $t0 = j * 4
add $t0, $s3, $t0	# $t0 = &Z[j]
lw $s3, 0($t0)	# $s3 = Z[j]
add $s1, $s2, $s3	# g = h + Z[j]

Given the C statement: *A[12] = h + A[5]*
What is the corresponding MIPS assembly language code? Assume the variable *h* is assigned to *$s2* and the base address of array *A* is in *$s3*.

h	A[0]
$s2	$s3
lw $t0, 20($s3)	# $t0 contains value in A[5]
add $t0, $s2, $t0	# $t0 = h + A[5]
sw $t0, 48($s3)	# store $t0 in A[12]

CPU Time	Clock Cycle Time	Instruction Count	Frequency	Spec Ratio	Speedup
t	τ	IC	f (Clock Cycle Time)	$\frac{Ref\ time}{Exec\ time}$	$\frac{(Exec\ time)old}{(Exec\ time)new}$

The results of the SPEC CPU2006 bzip2 benchmark running an AMD Barcelona has an *instruction count of 2.389e12*, an *execution time of 750s*, and a *reference time of 9650s*.

a) Find the change in CPU time if the number of instructions of the benchmark is increased by 10% and the CPI is increased by 5%.

$$t = 1.10(IC) * 1.05(CPI) \Rightarrow t = 1.155\ or\ 15.5\%$$

b) Find the change in the SPECratio if the number of instructions of the benchmark is increased by 10% and the CPI is increased by 5%.

$$SPECratio = \frac{9650}{750} \Rightarrow S.R. = 12.87$$

$$SPECratio = \frac{9650}{750 * 1.155} \Rightarrow S.R. = 11.14$$

$$Change = \frac{12.87 - 11.14}{12.87} \Rightarrow Change = 13.44\%$$

Assume the following for a *single 4 GHz* processor running a program:

Operation	# of instructions	CPI (Cycles per Instruction)
Arithmetic	1.28e9	3
Load/Store	2.56e9	8
Branch	64 million (=64e6)(=.064e9)	5

When the program is parallelized to run on parallel processors, the number of arithmetic and load/store instructions per processor is divided by *0.5p* (where p is the number of processors) but the number of branch instructions per processor remains the same. Find the **execution time** and **relative speedup** when the program is parallelized to run on *16* parallel processors.

$$(Exec\ time)old \Rightarrow t = \frac{\sum(IC * CPI)}{f} \Rightarrow \frac{(1.28e9 * 3) + (2.56e9 * 8) + (64e6 * 5)}{4e9} = \frac{24.64e9}{4e9} = 6.16s$$

$$(Exec\ time)new \Rightarrow t = \frac{\sum\left(\frac{IC}{0.5p} * CPI\right)}{f} \Rightarrow \frac{\left(\frac{1.28e9}{(0.5 * 16)} * 3\right) + \left(\frac{2.56e9}{(0.5 * 16)} * 8\right) + (64e6 * 5)}{4e9} = \frac{3.36e9}{4e9} = 840e - 3s$$

$$speedup = \frac{(Exec\ time)old}{(Exec\ time)new} \Rightarrow \frac{6.16}{840e - 3} = 7.33s$$

The following table shows results for SPEC benchmark programs running on a processor. Find the geometric mean.

Name	IC * 10e9	Execution Time (s)	Reference Time (s)	Ratio $\frac{Ref\ time}{Exec\ time}$
P1	336	1085	8993	$\frac{8993}{1085} = 8.288$
P2	1525	700	4960	$\frac{4960}{700} = 7.086$
P3	2300	820	1288	$\frac{1288}{820} = 1.571$

$$Geometric\ mean = \prod (SPECratio)^{\frac{1}{P}} \Rightarrow (8.288 * 7.086 * 1.571)^{\frac{1}{3}} = 4.519$$

Function	ALU Control	Funct Field	ALU Op	RegDst	ALUSrc	MemtoReg	RegWrite	MemRead	MemWrite	Branch
AND	0000	100100	10	1	0	0	1	0	0	0
OR	0001	100101	10	1	0	0	1	0	0	0
ADD	0010	100000	10	1	0	0	1	0	0	0
SUB	0110	100010	10	1	0	0	1	0	0	0
SLT	0111	101010	10	1	0	0	1	0	0	0
NOR	1100									
lw	0010	XXXXXX	00	0	1	1	1	1	0	0
sw	0010	XXXXXX	00	X	1	X	0	0	1	0
beq	0110	XXXXXX	01	X	0	X	0	0	0	1

lw $rt, offs($rs) → $rt=[Mem($rs+offs)]   sw $rt, offs($rs) → [Mem($rs+offs)]←$rt

Without Forwarding	With ALU-ALU Forwarding	With Full Forwarding
300ps	360ps	400ps

For lw, value is not ready until 1st half of WB, unless full-forw, then can be ready at 2nd half of MEM to be fed into 1st half of EX in next op. WB feeds ID

no-forwarding	1	2	3	4	5	6	7	8	9	
lw $s1, 40($s6)	IF	ID	EX	MEM	WB					
add $s6, $s2, $s2		IF	ID	EX	MEM	WB				
nop			*							
nop				*						
sw $s6, 50($s1)					IF	ID	EX	MEM	WB	t=300*9→2700ps

alu-alu forwarding	1	2	3	4	5	6	7	8	9	$speedup = \frac{(Exec\ time)old}{(Exec\ time)new} \Rightarrow \frac{2700}{3240} = 0.833$
lw $s1, 40($s6)	IF	ID	EX	MEM	WB					
add $s6, $s2, $s2		IF	ID	EX	MEM	WB				
nop			*							
nop				*						
sw $s6, 50($s1)					IF	ID	EX	MEM	WB	t=360*9→3240ps

full-forwarding	1	2	3	4	5	6	7	8	9	$speedup = \frac{(Exec\ time)old}{(Exec\ time)new} \Rightarrow \frac{2700}{3200} = 0.844$
lw $s1, 40($s6)	IF	ID	EX	MEM	WB					
add $s6, $s2, $s2		IF	ID	EX	MEM	WB				
nop			*							
sw $s6, 50($s1)				IF	ID	EX	MEM	WB		t=400*8→3200ps