NYU Tandon School of Engineering

Fall 2023, ECE 6913

Review Problems & Solutions

1. Assume for arithmetic, load/store, and branch instructions, a processor has CPIs of 1, 12, and 5, respectively. Also assume that on a single processor a program requires the execution of 2.56E9 arithmetic instructions, 1.28E9 load/store instructions, and 256 million branch instructions. Assume that each processor has a 2 GHz clock frequency.

Assume that, as the program is parallelized to run over multiple cores, the number of arithmetic and load/store instructions per processor is divided by $0.7 \times p$ (where p is the number of processors) but the *number of branch instructions per processor remains the same*

a. Find the *total execution time (ET)* for this program on *1, 2, 4, and 8 processors*, and show the relative speedup of the 2, 4, and 8 processors result relative to the single processor result.

Instruction Count [IC]

Arithmetic IC[2] = 2.56B / [0.7 x 2] = 1.83B

Arithmetic IC[4] = 2.56B / [0.7 x 4] = 0.91B

Arithmetic IC[8] = 2.56B / [0.7 x 8] = 0.46B

Load/Store IC[2] = 1.28B / [0.7 x 2] = 0.91B

Load/Store IC[4] = 1.28B / [0.7 x 4] = 0.46B

Load/Store IC[8] = 1.28B / [0.7 x 8] = 0.23B

Branch [1] =Branch [2] = Branch [4] = Branch [8] = 0.256B

Instruction	CPI	IC [1]	IC[2]	IC[4]	IC[8]
Arithmetic	1	2.56B	1.83B	0.91B	0.46B
Load/Store	12	1.28B	0.91B	0.46B	0.23B
Branch	5	0.256B	0.256B	0.256B	0.256B
Total Instruction Count [IC]		**4.096B**	**2.996B**	**1.626B**	**0.946B**

ET (Execution Time) = IC (Instruction Count) x CPI (Cycles per Instruction) x Cycle Time: 1/[2x10^9 Hz]

Instruction	CPI	ET [1]	ET [2]	ET [4]	ET [8]
Arithmetic	1	1x2.56Bx0.5ns=1.28s	1x1.83Bx.5n=0.91s	1x0.91Bx.5n=0.46s	1x0.46Bx0.5n=0.23s
Load/Store	12	12x1.28Bx.5ns=7.68s	12x0.91Bx.5ns=5.46s	12x0.46Bx.5n=2.76s	12x0.23Bx.5n=1.38s
Branch	5	5x 0.256Bx.5ns=0.64s	5x 0.256Bx.5ns=0.64s	5x 0.256Bx.5ns=0.64s	5x 0.256Bx.5ns=0.64s
Total	-	9.6s	7.01s	3.86s	2.17s

b. If the *CPI of the arithmetic instructions was doubled*, what would the impact be on the execution time of the program on 1, 2, 4, or 8 processors?

ET (Execution Time) = IC (Instruction Count) x CPI (Cycles per Instruction) x Cycle Time: 1/[2x10^9 Hz]

Instruction	**CPI**	**ET [1]**	**ET [2]**	**ET [4]**	**ET [8]**
Arithmetic	**2**	**2**x2.56Bx0.5ns=2.56s	**2**x1.83Bx.5n=1.83s	**2**x0.91Bx.5n=0.91s	**2**x0.46Bx0.5n=0.46s
Load/Store	12	12x1.28Bx.5ns=7.68s	12x0.91Bx.5ns=5.46s	12x0.46Bx.5n=2.76s	12x0.23Bx.5n=1.38s
Branch	5	5x 0.256Bx.5ns=0.64s	5x 0.256Bx.5ns=0.64s	5x 0.256Bx.5ns=0.64s	5x 0.256Bx.5ns=0.64s
Total	-	10.88s	7.93s	4.31s	2.48s

c. To what should the **CPI of load/store instructions be reduced** in order *for a single processor to match the performance of four processors* using the original CPI values?

If a single processor is to execute the same Program in **3.86 seconds** (instead of **9.6 seconds**), then the difference of **5.74 seconds** must come from the improvement in execution time of the *L/S instructions*. This improvement can be accomplished *by reducing L/S instruction CPI from 12 to CPIX*

7.68s-5.74s = 1.94s = *Time now available for single processor to process L/S instructions*

CPIX x 1.28Bx 0.5ns = 1.94s

So, CPIX = 1.94s/[1.28 x 0.5] = 1.94s/0.64 = **3.03**

2. The results of the SPEC CPU2006 bzip2 benchmark running on an AMD Barcelona has an instruction count of 2.389E12, an execution time of 750 s, and a reference time of 9650 s

a. Find the CPI if the clock cycle time is 0.333 ns.

IC = 2.389T, ET=750s, RefET = 9650s

CPI = (ET * clock rate)/IC = 750s x 3.00GHz / 2.389T = **0.94**

b. Find the SPECratio

SPECratio = T_ref/T = 9650s /750s = **12.86**

c. Find the increase in CPU time if the number of instructions of the benchmark is increased by 10% without affecting the CPI

ET = CPI x IC x Tcycle = 0.94 x 2.389T x 1.1 x 0.333ns = 822s or an **increase in ET of 9.67%**

d. Find the increase in CPU time if the number of instructions of the benchmark is increased by 10% and the CPI is increased by 5%.

ET = CPI x IC x Tcycle = 0.94 x 1.05 x 2.389T x 1.1 x 0.333ns = 863s or an **increase in ET of 15%**

e. Find the change in the SPECratio for this change

SPECratio = T_ref/T = 9650s /863s = **11.18**

f. Suppose that we are developing a new version of the AMD Barcelona processor with a **4 GHz clock rate**. We have added some additional instructions to the instruction set in such a way that *the number of instructions has been reduced by 15%*. The *execution time is reduced to 700 s* and the new SPECratio is 13.7. Find the new CPI.

ETnew = 700s = CPIX x 2.389T x 0.85 x 0.250ns

So, CPIX = 700/507.66 = **1.38**

g. This CPI value is larger than obtained in part a. as the clock rate was increased from 3 GHz to 4 GHz. Determine whether the increase in the CPI is similar to that of the clock rate. If they are dissimilar, why?

CPI increased by 1.38/0.94 = 1.46

Clock rate increased by 4/3 = 1.33

Both – Instruction Count *and* the Clock Cycle time were lowered as a result of adding instructions to the instruction set.

Since each instruction, on average, is accomplishing more work, the reduced cycle time is likely to require the average instruction to complete its task using more cycles and hence an increase in the average CPI of the Bench mark

h. By how much has the CPU time been reduced?

700s / 750s = 0.933 or **by 6.67%**

3. Assume a program requires the execution of 50 $\times 10^6$ FP instructions, 110 $\times 10^6$ INT instructions, 80 $\times 10^6$ L/S instructions, and 16 $\times 10^6$ branch instructions. The CPI for each type of instruction is 1, 1, 4, and 2, respectively. Assume that the processor has a 2 GHz clock rate.

a. By how much must we improve the CPI of FP instructions if we want the program to run two times faster?

Clock cycles = $\mathbf{CPI_{fp}}$ × #FP instr. + $\mathbf{CPI_{int}}$ × #INT instr. + $\mathbf{CPI_{l/s}}$ × #L/S instr. + $\mathbf{CPI_{branch}}$ × #branch instr.

$= \mathbf{1}\text{x}50\times10^6 + \mathbf{1}\text{x}110\times10^6 + \mathbf{4}\text{x}80\times10^6 + \mathbf{2}\text{x}16\ \times10^6$
$= 512\ \times\ 10^6$

If we lower the number of clock cycles by improving CPIFP, even if the CPI of FP instructions goes to zero, the program will still consume more than half of the 512M clock cycles with all 4 instruction classes: 292M cycles. **So, the program cannot possibly run 2 times faster by improving CPI of FP instructions**

b. By how much must we improve the CPI of L/S instructions if we want the program to run two times faster?

Number of cycles consumed by FP, INT and Branch instructions = 192M

To lower the total number of cycles (512M by half → 256M cycles), the number of L/S cycles must lower to a max of 256M – 192M = 64M cycles. Since the number of L/S instructions equals 80M, the $\text{CPIX}_{L/S}$ must reduce to $\mathbf{CPIX_{L/S}}$ **= 64M cycles/ 80M Instructions = 0.8**

c. By how much is the execution time of the program improved if the CPI of INT and FP instructions is reduced by 40% and the CPI of L/S and Branch is reduced by 30%?

With Initial assumptions of CPIX for each instruction class:

Clock cycles = $\mathbf{CPI_{fp}}$ × #FP instr. + $\mathbf{CPI_{int}}$ × #INT instr. + $\mathbf{CPI_{l/s}}$ × #L/S instr. + $\mathbf{CPI_{branch}}$ × #branch instr.

$= \mathbf{1}\text{x}50\times10^6 + \mathbf{1}\text{x}110\times10^6 + \mathbf{4}\text{x}80\times10^6 + \mathbf{2}\text{x}16\ \times10^6$
$= \mathbf{512\ \times\ 10^6}$

Clock Cycles with Improved CPIX for INT, FP, L/S:

Clock cycles = $\mathbf{CPI_{fp}}$ × #FP instr. + $\mathbf{CPI_{int}}$ × #INT instr. + $\mathbf{CPI_{l/s}}$ × #L/S instr. + $\mathbf{CPI_{branch}}$ × #branch instr.

$= \mathbf{0.6}\text{x}50\times10^6 + \mathbf{0.6}\text{x}110\times10^6 + \mathbf{2.8}\text{x}80\times10^6 + \mathbf{1.4}\text{x}16\ \times10^6$
$= \mathbf{404\ \times\ 10^6}$

that is a speedup of 512/404 = 26.73%

4. When a program is adapted to run on multiple processors in a multiprocessor system, the execution time on each processor is comprised of computing time and the overhead time required for locked critical sections and/or to send data from one processor to another. Assume a program requires t =100 s of execution time on one processor. When run p processors, each processor requires t/p s, as well as an additional 4 s of overhead, irrespective of the number of processors.

Compute the per-processor execution time for 2, 4, 8, 16, 32, 64, and 128 processors. For each case, list the corresponding speedup relative to a single processor and the ratio between actual speedup versus ideal speedup (speedup if there was no overhead).

# Processors	Ex Time/Processor	Time w/ overhead	Speedup	Actual Speedup/ideal speedup
1	100			
2	50	54	100/54 = 1.85	1.85/2 = 0.93
4	25	29	100/29 = 3.44	3.44/4 = 0.86
8	12.5	16.5	100/16.5 = 6.06	6.06/8 = 0.75
16	6.25	10.25	100/10.25 = 9.76	9.76/16 = 0.61

5. The Pentium 4 Prescott processor, released in 2004, had a clock rate of 3.6 GHz and voltage of 1.25 V. Assume that, on average, it consumed 10 W of static power and 90 W of dynamic power.

The Core i5 Ivy Bridge, released in 2012, has a clock rate of 3.4 GHz and voltage of 0.9 V. Assume that, on average, it consumed 30 W of static power and 40 W of dynamic power.

a. For each processor find the average capacitive loads.

½ CV^2F = ½ $C(1.25)^2$x3.6 GHz = 90 W [Prescott] => C = 32nF

½ CV^2F = ½ $C(0.9)^2$x3.4 GHz = 40 W [Ivy Bridge] => C = 29nF

b. Find the percentage of the total dissipated power comprised by static power and the ratio of static power to dynamic power for each technology

Prescott: fraction of total power from Static dissipation = 10/100 = 10%

Ivy Bridge: 30/70 = 42.9%

c. If the *total dissipated power is to be reduced by 10%, how much should the voltage be reduced* to maintain the same leakage current? Note: power is defined as the product of voltage and current.

Total Power new / Total Power old = 0.9 (1a)

or, (Dn + Sn) / (Do + So) = 0.9 (1b)

Dynamic Power new = Dn = C V_{new}^2 x F (2)

Static Power new = Sn = V_{new} x I_L (3)

Static Power old = So = V_{old} x I_L (4)

For the Prescott:

From (2) V_{new} = sqrt (Dn/[C x F]) =

$$\sqrt{\frac{D_n}{32nF \times 3.6GHz}} = \sqrt{\frac{D_n}{115.2}} \qquad (5)$$

From (1b) Dn = 0.9 (So + Do) – Sn (6)

From (3), (4), Sn = So(V_{new}/V_{old}) = (7)

Sn = V_{new} x (10W/1.25V) = V_{new} x 8 (8)

From (6), Dn = 0.9 x 100W – 8 V_{new}

So, Dn = [90-8V_{new}] (9)

From (5), (9) $V_{new} = \sqrt{\frac{90-8V_{new}}{32nF \times 3.6GHz}} =$

$$V_{new} = \sqrt{\frac{90-8V_{new}}{115.2}} \qquad (10)$$

Solving above quadratic for V_{new}, we get

$V_{new} = 0.85V$ (11)

So, for the Prescott, operating voltage would scale down from 1.25V to 0.85V

6. Assume that for a given program 70% of the executed instructions are arithmetic, 10% are load/store, and 20% are branch.

a. Given this instruction mix and the assumption that an arithmetic instruction requires two cycles, a load/store instruction takes six cycles, and a branch instruction takes three cycles, find the average CPI.

CPI of program = weighted sum of instruction mix

CPI = 0.7 x 2 + 0.1 x 6 + 0.2 x 3 = 2.6

b. For a 25% improvement in performance, how many cycles, on average, may an arithmetic instruction take if load/store and branch instructions are not improved at all?

For a 25% improvement, we must reduce the CPI to 2.6 * 0.75 = 1.95 cycles

Thus, we want 0.7 * **x** + 0.1 * 6 + 0.2 * 3 < = 1.95. Solving for x shows that the arithmetic instructions must have a CPI of at most **x=1.07**

c. For a 50% improvement in performance, how many cycles, on average, may an arithmetic instruction take if load/store and branch instructions are not improved at all?

For a 50% improvement, we must reduce the CPU to 2.6*.5 = 1.3.

Thus, we want 0.7***x** + 0.1*6 + 0.2*3 < = 1.3

Solving for x shows that the arithmetic instructions must have a CPI of at most **x=0.14**

7. Assume for a given processor the CPI of *arithmetic* instructions is 1, the CPI of *load/store* instructions is 10, and the CPI of *branch* instructions is 3. Assume a program has the following instruction breakdowns: 500 million *arithmetic* instructions, 300 million *load/store* instructions, 100 million *branch* instructions.

a. Suppose that new, more powerful *arithmetic* instructions are added to the instruction set. On average, through the use of these more powerful *arithmetic* instructions, we can *reduce the number* of *arithmetic* instructions needed to execute a program by 25%, while *increasing the clock cycle time* by only 10%. Is this a good design choice? Why?

No. The resulting machine would be slower overall.

Current CPU requires (# arithmetic * 1 cycle) + (# load/store * 10 cycles) + (# branch/jump * 3 cycles) = 500*1 + 300*10 + 100*3 = 3800 cycles.

The new CPU requires (.75* # arithmetic * 1 cycle) + (# load/store * 10 cycles) + (# branch/jump * 3 cycles) = 375*1 + 300*10 + 100*3 = 3675 cycles.

However, given that each of the new CPU's cycles is 10% longer than the original CPU's cycles, the new CPU's 3675 cycles will take as long as 4042.5 cycles on the original CPU

b. Suppose that we find a way to double the performance of arithmetic instructions. What is the overall speedup of our machine? What if we find a way to improve the performance of arithmetic instructions by 10 times?

If we double the performance of arithmetic instructions by reducing their CPI to 0.5, then the CPU will run the reference program in (500*.5) + (300*10) + 100*3 = 3550 cycles. This represents a speedup of 1.07.

If we improve the performance of arithmetic instructions by a factor of 10 (reducing their CPI to 0.1), then the CPU will run the reference program in (500*.1) + (300*10) + 100*3 = 3350 cycles. This represents a speedup of 1.13.

8 a. Provide the instruction type and assembly language instruction for the following binary value:

0000 0000 0001 0000 1000 0000 1011 0011$_2$

```
R -type: add x1, x1, x1
```

8 b. Provide the instruction type and hexadecimal representation of the following instruction:

```
sd x5, 32(x30)
```

8 c. Provide the instruction type, assembly language instruction, and binary representation of instruction described by the following RISC-V fields:

```
opcode=0x33, funct3=0x0, funct7=0x20, rs2=5, rs1=7, rd=6
```

```
R -type: sub x6, x7, x5
in Hex: 0x40538333:  0100 0000 0101 0011 1000 0011 0011 0011
```

9. Assume that **`x5`** holds the value 128_{10}.

a. For the instruction **`add x30, x5, x6,`** what is the range(s) of values for **`x6`** that would result in overflow?

There is an overflow if $128 + x6 > 2^{63} - 1$
In other words, if $x6 > 2^{63} - 129$
There is also an overflow if $128 + x6 < -2^{63}$
that is, if $x6 < -2^{63} - 128$ (which is impossible given the range of x6)

b. For the instruction **`sub x30, x5, x6,`** what is the range(s) of values for **`x6`** that would result in overflow?

There is an overflow if $128 - x6 > 2^{63} - 1$
In other words, if $x6 < -2^{63} + 129$
There is also an overflow if $128 - x6 < -2^{63}$
In other words, if $x6 > 2^{63} + 128$ (which is impossible given the range of x6).

c. For the instruction **`sub x30, x6, x5,`** what is the range(s) of values for **`x6`** that would result in overflow?

There is an overflow if $x6 - 128 > 2^{63} - 1$
In other words, if $x6 < 2^{63} + 127$ (which is impossible given the range of x6)
There is also an overflow if $x6 - 128 < -2^{63}$
In other words, if $x6 < -2^{63} + 128$

10. Write out instruction(s) to **copy contents at one memory location to another:** `B[g+2] = A[i+j+4]`. Assume `i, j, g` values are in registers `x5, x6, x7`. Assume base address in memory of Array data structures `'A, B'` (or address in memory of '`A[0]`' and '`B[0]`') are stored in Registers `x28, x29`

`[very similar to HW 2 problems]`

11. The `Hewlett-Packard 2114, 2115, and 2116` used a format with the leftmost 16 bits being the fraction stored in *two's complement format*, followed by another 16-bit field which had the leftmost 8 bits as an extension of the fraction (making the fraction 24 bits long), and the *rightmost 8 bits representing the exponent*. However, in an interesting twist, *the exponent was stored in sign-magnitude format with the sign bit on the far right*!

Write down the bit pattern to represent -1.5625×10^{-1} assuming this format. *No hidden 1 is used*. Comment on how the range and accuracy of this 32-bit pattern compares to the single precision IEEE 754 standard.

```
-1.5625 × 10^-1 = - 0.15625 × 10^0
= -0.00101 × 2^0
move the binary point two positions to the right
= -0.101 × 2^-2

The (absolute value of the) fraction is 0101 ( we must write the
leading zero to represent 2s complement sign)

0101 0000 0000 0000 0000 0000 [24 bits for fraction field -
representing the absolute value of the fraction: 0.101]

Taking the 2s complement - to negate this 24 bit number (since the
fraction is negative = -0.101)

Step 1: reverse the bits
 1010 1111 1111 1111 1111 1111
Step 2: add 1
 1010 1111 1111 1111 1111 1111
+0000 0000 0000 0000 0000 0001
-------------------------------
 1011 0000 0000 0000 0000 0000

exponent = -2,
using the 8 bits for the exponent field with right most bit
corresponding to the sign bit:  0000 0101

Thus, 32-bit string: 10110000000000000000000000000101
```

12. IEEE 754-2008 contains a half precision that is only 16 bits wide. The leftmost bit is still the sign bit, the exponent is 5 bits wide and has a *bias of 15*, and the mantissa is *10 bits long*. A hidden 1 is assumed. Write down the bit pattern to represent -1.5625×10^{-1} assuming a version of this format, which uses an excess-16 format to store the exponent. Comment on how the range and accuracy of this 16-bit floating point format compares to the single precision IEEE 754 standard.

$-1.5625 \times 10^{-1} = - 0.15625 \times 10^{0}$
$= -0.00101 \times 2^{0}$

move the binary point three positions to the right, =

-1.01×2^{-3}

exponent = − 3
exponent field = exponent + bias =
− 3 + 15 = 12, expressed in the 5 exponent field bits as:
01100

fraction = **−** 0.**0100000000**

16-bit string:
1011000100000000