EE466 Fall 2019 HW 1

Lewis Collum (0621539) EE/CE

Due: September 18, 2019

1.2

Letter	Matching Idea
a	Performance via Pipelining
b	Dependability via Redundancy
c	Performance via Prediction
d	Performance via Parallelism
e	Hierarchy of Memories
f	Make the Common Case Fast
g	Design for Moore's Law
h	Use Abstraction to Simplify Design

1.5

```
import pint
from collections import namedtuple

unit = pint.UnitRegistry()
unit.define('cycles=')
unit.define('instructions=')

Processor = namedtuple('Processor', ['cyclesPerSecond', 'cyclesPerInstruction'])
processors = {
    'p1': Processor(3e9 * unit.cycles/unit.seconds, 1.5*unit.cycles/unit.instructions),
    'p2': Processor(2.5e9 * unit.cycles/unit.seconds, 1.5*unit.cycles/unit.instructions),
    'p3': Processor(4e9 * unit.cycles/unit.seconds, 1.5*unit.cycles/unit.instructions)}

programExecutionTime = 10 * unit.seconds
for name, processor in processors.items():
    performance = processor.cyclesPerSecond / processor.cyclesPerInstruction
    instructions = performance * programExecutionTime
    cycles = processor.cyclesPerSecond * programExecutionTime
    newClockRate = instructions*processor.cyclesPerInstruction*(1+0.2) / (programExecution
```

```python
    print((f"{name}:\n"
           f"  performance = {performance:.2E}\n"
           f"  instructions (#) = {instructions:.2E}\n"
           f"  cycles (#) = {cycles:.2E}\n"
           f"  new clock rate for 30% reduction in execution time = {newClockRate:.2E}\n"))
```

```
p1:
  performance = 2.00E+09 instructions / second
  instructions (#) = 2.00E+10 instructions
  cycles (#) = 3.00E+10 cycles
  new clock rate for 30% reduction in execution time = 5.14E+09 cycles / second

p2:
  performance = 1.67E+09 instructions / second
  instructions (#) = 1.67E+10 instructions
  cycles (#) = 2.50E+10 cycles
  new clock rate for 30% reduction in execution time = 4.29E+09 cycles / second

p3:
  performance = 2.67E+09 instructions / second
  instructions (#) = 2.67E+10 instructions
  cycles (#) = 4.00E+10 cycles
  new clock rate for 30% reduction in execution time = 6.86E+09 cycles / second
```

a.

p3 has the highest performance at 2.67×10^9 instructions/second.

b.

refer to code results

c.

refer to code results

1.8.1

Equation ID	Processor Name	Clock Rate	Voltage	Dynamic Power
1	Pentium 4 Prescott Processor	3.6GHz	1.25V	90W
2	Core i5 Ivy Bridge	3.4GHz	0.9V	40W

$$P = \frac{1}{2}CV^2F$$
$$\implies C = \frac{2P}{V^2F}$$
$$C_1 = \frac{2 \cdot 90}{1.25^2 \cdot 3.6 \times 10^9} = 3.2 \times 10^{-8}F$$
$$C_2 = \frac{2 \cdot 40}{0.9^2 \cdot 3.4 \times 10^9} = 2.9 \times 10^{-8}F$$

1.10.1 & 1.10.2

```python
from collections import namedtuple
import pint
import math

def areaFromDiameter(diameter: float) -> float:
    radius = diameter/2
    return math.pi*radius**2

unit = pint.UnitRegistry()
unit.define('defects=')

Wafer = namedtuple('Wafer', ['diameter', 'cost', 'dieCount', 'defectRatio'])
wafers = {
    'w1': Wafer(
        diameter = 15 * unit.cm,
        cost = 12,
        dieCount = 84,
        defectRatio = 0.020 * unit.defects/unit.cm**2),
    'w2': Wafer(
        diameter = 20 * unit.cm,
        cost = 15,
        dieCount = 100,
        defectRatio = 0.031 * unit.defects/unit.cm**2)
}

for name, wafer in wafers.items():
    waferArea = areaFromDiameter(wafer.diameter)
```

```python
    dieArea = waferArea/wafer.dieCount

    waferYield = 1/(1+(wafer.defectRatio * dieArea/2))**2
    costPerDie = wafer.cost / (wafer.dieCount * waferYield)

    print((f"{name}:\n"
           f"  yield = {waferYield.magnitude:.3}\n"
           f"  cost per die = {costPerDie.magnitude:.3}\n"))
```

```
w1:
  yield = 0.959
  cost per die = 0.149

w2:
  yield = 0.909
  cost per die = 0.165
```

1.11

```python
instructions = 2.389e12
executionTime = 750

cycleTime = 0.333e-9
cycleRate = 1/cycleTime

instructionRate = instructions/executionTime
cyclesPerInstruction = cycleRate/instructionRate
print(f"1.11.1) CPI = {cyclesPerInstruction:.2}")

referenceTime = 9650
specRatio = referenceTime / executionTime
print(f"1.11.2) SPECratio = {specRatio:.3}")

print(f"1.11.3) CPU time = {(1+0.1)*instructions * cyclesPerInstruction / cycleRate} secon
print(f"1.11.4) CPU time = {(1+0.1)*instructions * (1+0.5)*cyclesPerInstruction / cycleRate
```

```
1.11.1) CPI = 0.94
1.11.2) SPECratio = 12.9
1.11.3) CPU time = 825.0 seconds
1.11.4) CPU time = 1237.5 seconds
```

1.12.1

```python
from collections import namedtuple
import pint
unit = pint.UnitRegistry()
unit.define('cycles=')
unit.define('instructions=')

Processor = namedtuple('Processor', ['cycleRate', 'cyclesPerInstruction', 'instructions'])
processors = {
    'P1': Processor(
        cycleRate = 4e9 * unit.cycles/unit.seconds,
        cyclesPerInstruction = 0.9 * unit.cycles/unit.instructions,
        instructions = 5e9 * unit.instructions),
    'P2': Processor(
        cycleRate = 3e9 * unit.cycles/unit.seconds,
        cyclesPerInstruction = 0.75 * unit.cycles/unit.instructions,
        instructions = 1e9 * unit.instructions)
}

for name, processor in processors.items():
    cpuTime = processor.instructions * processor.cyclesPerInstruction / processor.cycleRate
    print((f"{name}: \n"
           f"  Clock Rate = {processor.cycleRate:.1E}\n"
           f"  Performance = {1/cpuTime:.2}\n"))
```

```
P1:
  Clock Rate = 4.0E+09 cycles / second
  Performance = 0.89 / second

P2:
  Clock Rate = 3.0E+09 cycles / second
  Performance = 4.0 / second
```

$\boxed{\text{P1 has a higher clock rate and a lower performance.}}$

1.13

```python
fpTime = 70
lsTime = 85
branchTime = 40
intTime = 55
programTime = fpTime + lsTime + branchTime + intTime

print(f"1.13.1) {fpTime - (1-0.2)*fpTime} seconds reduced for total time")
```

```
print(f"1.13.2) {intTime - ((1-0.2)*programTime-fpTime-lsTime-branchTime)} seconds reduced
print(f"1.13.3) {((1-0.2)*programTime-fpTime-lsTime-intTime)} seconds left for branching")
```

```
1.13.1) 14.0 seconds reduced for total time
1.13.2) 50.0 seconds reduced for INT
1.13.3) -10.0 seconds left for branching
```

Can the total time can be reduced by 20% by reducing only the time for branch instructions? **No, there would be negative time left for branching (as shown above).**