Solver: Dang Xuan Vu

Email Address: xuanvu001@e.ntu.edu.sg

1.

a) Assumed that the program comprises of n instructions with execution time of 1 unit
Original execution time: $n \; x \; 1 = n \; (units)$
Enhanced execution time: $\frac{n}{2} x \frac{1}{10} + \frac{n}{2} x \frac{1}{5} = \frac{3n}{20} (units)$

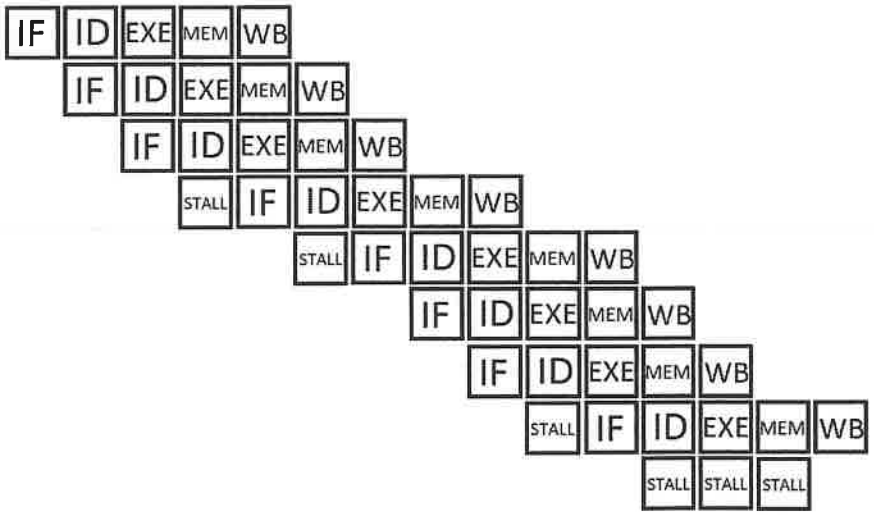
➔ $Speedup = \frac{Original \; exe. \; time}{Enhanced \; exe. \; time} = \frac{20}{3} \sim 6.67$

b)

(i) Types of hazards:
- Data hazard
- Control hazard

(ii) Assume that data is needed at ID stage and PC update for branch is after the MEM stage. According to the diagram, there are 6 stall cycles for each loop

```
        addi  $7, $0, 0
        addi  $6, $0, 36
loop    lw $10, 0($5)
        add   $7, $4, $10
        sw$7, 0($5)
        addi  $6, $6, -1
        addi  $5, $5, 4
        bgtz  $6, loop
finish
```



(iii) Code segment with loop unrolling:

```
        addi  $7, $0, 0
        addi  $8, $0, 0
        addi  $6, $0, 18
loop    lw    $10, 0($5)
        lw    $11, 1($5)
        add   $7, $4, $10
        add   $8, $4, $11
        addi  $6, $6, -1
        sw    $7, 0($5)
        bgtz  $6, loop
        addi  $5, $5, 4
        sw    $8, 1($5)
finish
```

➔ Total reduction of stall cycles: 5 cycles/loop

c) Heat sink is a passive heat exchanger that maximizes the contact area of the electronic device with a coolant fluid such as air. Heat sink are used to support the heat dissipation of the basic device when the device itself cannot moderate its temperature.

2.

a) Structure of a JUMP instruction is:

$$[opcode] :: [sign\ bit] :: [offset] - [31:26] :: [25] :: [24:0]$$

➔ Maximum forwarding in byte is 0x1FFFFFF = $2^{25} - 1$ (bytes)

Since each instruction word comprises of 4-bytes (32-bit), maximum forwarding in instruction is: $(2^{25} - 1) / 4 = 2^{23} - 1$ (instructions)

b)

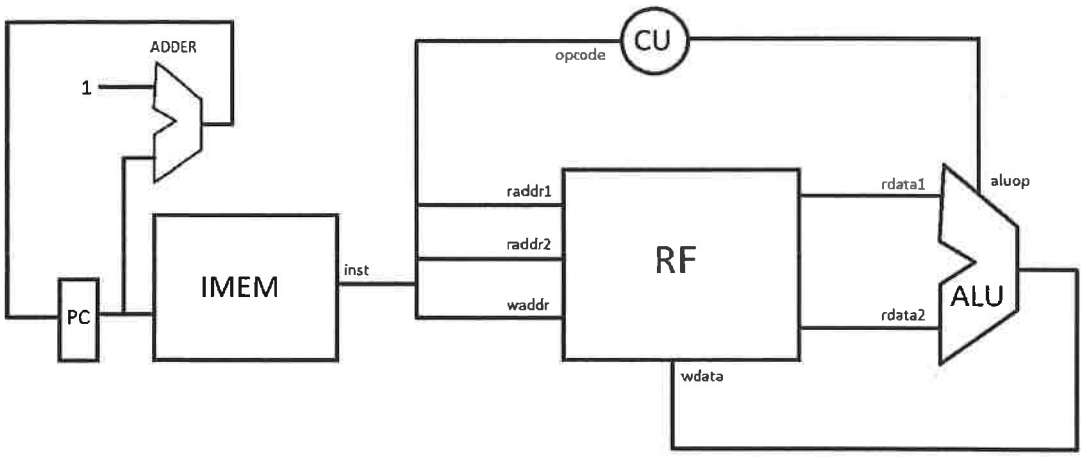
(i) For ADDI instructions, the offset is sign-extended from 1001_2 (9) into 0xFFFFFFF9, so the content of register $15 after execution is:

$$0x7FFFFFF7 + 0xFFFFFFF9 = 0x7FFFFFF0$$

(ii) For ADDIU instructions, the offset is zero-extended from 1001_2 (9) into 0x00000009, so the content of register $15 after execution is:

$$0x7FFFFFF7 + 0x00000009 = 0x80000000$$

c) R-format ALU instructions datapath



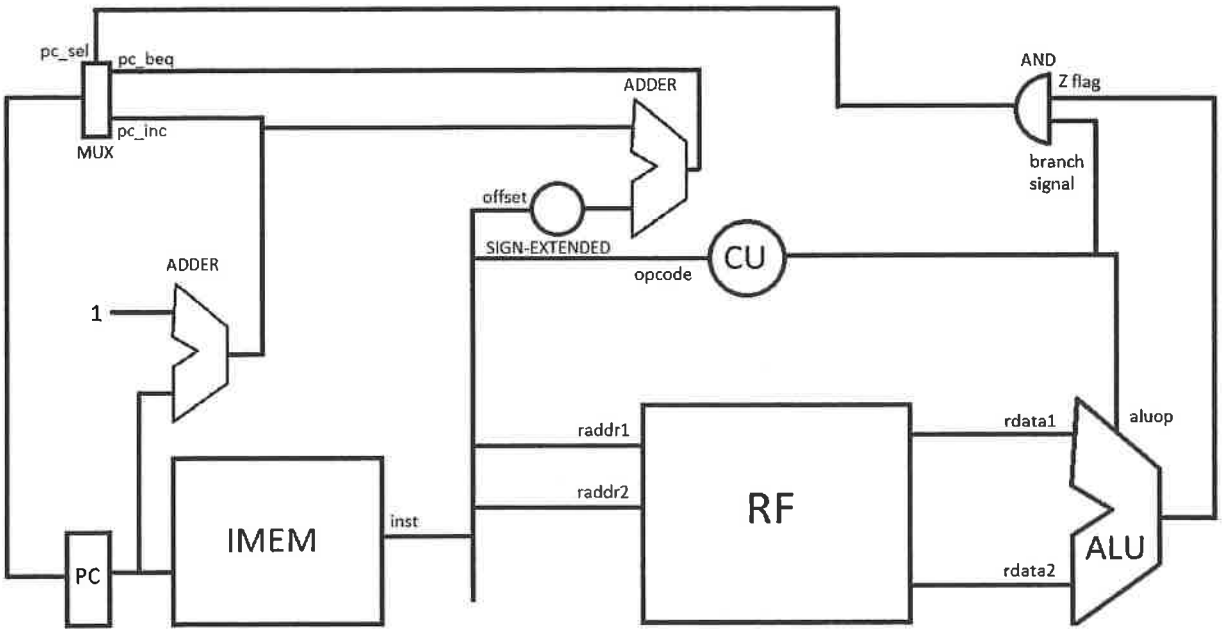
LEGEND:

PC: Program Counter
IMEM: Instruction Memory
ALU: Arithmetic – Logic Unit
RF: Register File
CU: Control Unit

To derive a datapath for Load/Store instructions from that of R-format ALU instructions, 3 MUXes are required:

- 1 MUX to choose the destination register for the Register File ($rd/$rt)
- 1 MUX to choose the 2nd operand for the ALU ($rt/imm)
- 1 MUX to choose the value source for Register File writeback (ALU/DM)

d) I-format BEQ instructions datapath



LEGEND:

PC: Program Counter
IMEM: Instruction Memory
ALU: Arithmetic – Logic Unit
RF: Register File
CU: Control Unit

Execution of BRANCH instructions:

- Instruction is fetched & decoded in the first two stages
- ALU compares the content of $rs & $rt
- The offset is left-shifted by 2 to convert byte-addressable to word-addressable
- If the result is equal ($rs == $rt), PC = PC (incremented) + offset (sign-extended)
- Otherwise if not equal, PC = PC (incremented)

3.

a)

(i) Prediction Accuracy = 4/12 = 33.33%

Branch Prediction	Branch Outcome	Hit/Miss
N	T	M
T	T	H
T	T	H
T	N	M
N	T	M
T	N	M
N	T	M
T	T	H
T	T	H
T	N	M
N	T	M
T	N	M

(ii) Methods to eliminate control hazards:

- Traditional
  - Stall the CPU for as many cycles as the branch instruction requires for PC updating, with minimum cost of 2 stall cycles
- Branch Prediction
  - Predict the outcome of the branch then compare with actual result for precision, can either be static (taken/not taken) or dynamic (multiple-bit predictor)
- Delayed Branching
  - Apply out-of-order execution to insert independent instructions subsequent to control instructions to minimize waste cycles

b) Parallelism Features:

- Instruction-level (Superscalar/VLIW processors)
  - Multiple independent instructions are identified and grouped to be executed concurrently in different functional units in a single processor
- Data-level (Vector/array processors)
  - Same operation is performed on multiple data values concurrently in multiple processing units
- Thread-level (Multi-core/processor systems)
  - More than one independent threads/tasks are executed simultaneously

c) Motivation of using GPU for high performance GPP computing:

- Data-level parallelism by massive data parallel stream processing
- Thread-level parallelism by using thousands of threads running in hundreds of cores
- High cost-effective due to GPU high performance
- Utilizes the already available hardware

Behavior of GPU compared with GPP:

- High latency & throughput vs. low latency & throughput

d) Relation between operating voltage & clock frequency:

$$f \propto V$$

when the threshold voltage (Vth) is negligible compared to the operating voltage (the full formula can be found in *p7, Module 1.4 Power dissipation in processors*).
Dynamic power consumption formula:

$$P_{dynamic} = ACV^2 f \propto ACf^3$$

where A: switching factor (workload equivalent)
C: total load capacitance
V: operating voltage
f: clock frequency

Compare to quad-core running at frequency f, a quad-core processor running at frequency f/4 has equal switching factor ($A_f = A_{f/4}$) and same chip area, or total capacitance ($C_f = C_{f/4}$). Dynamic power consumption in comparison:

$$P_{\frac{f}{4}} = A_{\frac{f}{4}} C_{\frac{f}{4}} (\frac{1}{4} f)^3 = \frac{1}{64} A_f C_f f^3 = \frac{1}{64} P_f = \frac{1}{16} P_{single}$$

Hence, for similar performance, using quad-core processors with sufficient independent threads available can greatly reduce the dynamic power consumption by 16 times compared to normal single-core processors.

4.

a)

(i) Address Format for Direct Mapped Cache:

$$[tag] :: [index] :: [offset] - [7:5] :: [4:2] :: [1:0]$$

Address Reference	Tag	Index	Offset
0x12	000	100	10
0x01	000	000	01
0x09	000	010	01
0x05	000	001	01
0x22	001	000	10
0x28	001	010	00
0x05	000	001	01
0x01	000	000	01
0x0A	000	010	10
0x21	001	000	01

Address Format for 4-way Set Associative Cache:

$$[tag] :: [index] :: [offset] - [7:3] :: [2] :: [1:0]$$

Address Reference	Tag	Index	Offset
0x12	00010	0	10
0x01	00000	0	01
0x09	00001	0	01
0x05	00000	1	01
0x22	00100	0	10
0x28	00101	0	00
0x05	00000	1	01
0x01	00000	0	01
0x0A	00001	0	10
0x21	00100	0	01

(ii) Cache Content for Direct Mapped Cache:

Index	Content
000	000 001 000 001
001	000
010	000 001 000
011	
100	000
101	

110	
111	

➔ Hit-rate = 1/10 = 10 %

Cache Content for 4-way Set Associative Cache:

Index	Way-1	Way-2	Way-3	Way-4
0	00010 00101	00000	00001	00100
1	00000			

➔ Hit-rate = 4/10 = 40%

(iii) Cache Content for Fully Associative Cache:

Way-1	Way-2	Way-3	Way-4	Way-5	Way-6	Way-7	Way-8
000100	000000	000010	000001	001000	001010		

➔ Hit-rate = 4/10 = 40%

➔ The Fully Associative scheme does not show any advantage compared with the 4-way Set Associative, as the memory content for latter reference stay in the cache without being cached out

b) Number of page table entries:

$$\frac{Virtual\ memory\ space}{Page\ size} = \frac{2^{16}}{64} = 2^{10}\ (entries)$$

Single-level page table size:

$$Number\ of\ entries\ x\ Entry\ size = 2^{10}\ (entries)\ x\ 4\ \left(\frac{B}{entry}\right) = 2^{12}B = 4MB$$

For reporting of errors and errata, please visit pypdiscuss.appspot.com
Thank you and all the best for your exams! ☺