USE YOUR IMAGINATION™

Blue Book

EXAMINATION BOOK

BOX NO. _____

NAME RAMAN KUMAR JHA (rj2712)

SUBJECT CSA

CLASS _____

SECTION A

INSTRUCTOR AZEEZ

DATE 10/02/2025

11" x 8.5" 8 LEAVES 16 PAGES

ROARING SPRING® PAPER PRODUCTS
ROARING SPRING, PA 16673

RAMAN KUMAR JHA
(rj2712)

① [a] Assuming the 4 byte word is stored in x5.

```
andi x6, x5, 0x00FF0000 #extract byte3
srli x6, x6, 8 # shift extracted byte3 to bits 15:8
andi x7, x5, 0x0000FF00 # extract byte2
slli x7, x7, 8 # shift extracted byte2 to bits 23:16
or x8, x6, x7 # combine swapped byte 2 and 3
andi x5, x5, 0xFF0000FF #Zero out bits
                        23:8 in original register
or x5, x5, x8 # load swapped byte 2 and
                3 in original register.
```

① [b] i) f = g - A[B[C[16]]]

```
lw x7, 64(x7)      # C[16]
slli x7, x7, 2
add x6, x6, x7     # &B[C[16]]
lw x6, 0(x6)       # B[C[16]]
slli x6, x6, 2
add x5, x5, x6     # &A[B[C[16]]]
lw x5, 0(x5)       # A[B[C[16]]]
sub x8, x9, x5     # f = g - A[B[C[16]]]
```

RAMAN KUMAR JHA
(rj2712)

① [b] (ii) $f = g - A[c[8] + B[64]]$

```
lw   x6, 256(x6)      # B[64]
lw   x7, 37(x7)       # c[8]
add  x6, x6, x7       # B[64] + c[8]
slli x6, x6, 2
add  x5, x5, x6       # &A[B[64] + c[8]]
lw   x5, 0(x5)        # A[B[64] + c[8]]
sub  x8, x9, x5       # f = g - A[B[64] + c[8]]
```

① [b] (iii) $A[i] = 8B[4i-41] + 8c[64i+64]$

```
slli x11, x10, 6      # 64i
addi x11, x11, 64     # 64i + 64
slli x11, x11, 2
add  x7, x7, x11      # &c[64i+64]
lw   x7, 0(x7)        # c[64i+64]
slli x7, x7, 3        # 8c[64i+64]
slli x12, x10, 2      # 4i
addi x12, x12, -41    # 4i - 41
slli x12, x12, 2
add  x6, x6, x12      # &B[4i-41]
lw   x6, 0(x6)        # B[4i-41]
slli x6, x6, 3        # 8B[4i-41]
add  x6, x6, x7       # 8c[64i+64] + 8B[4i-41]
slli x13, x10, 2
add  x5, x5, x13      # &A[i]
sw   x6, 0(x5)        # A[i] = 8B[4i-41] + 8c[64i+64]
```

1[c] $0x2F_{16} = 0010\ 1100 = 46_{10}$
$0x2D_{16} = 0011\ 1101 = 61_{10}$

$46 \times 61 = 2806_{10} = 0$

$0x2F_{16} = 0010\ 1110_2 = 46_{10}$, and
$46 = 32+8+4+2 = 2^5+2^3+2^2+2^1$

So,
we can shift 0x3D left 5 places
$= 0\ 0111\ 1010\ 0000 = 1952_{10} = 0x7A0_{16}$

then add 0x3D left shifted 3 places
$= 001\ 1100\ 1000 = 488_{10} = 0x1E8_{16}$

then add, 0x3D left shifted 2 places,
$= 00\ 111\ 1\ 0100 = 244_{10} = 0xF4_{16}$

then add 0x3D left shifted 1 place
$= 0\ 0111\ 1010 = 122_{10} = 0x7A_{16}$

Adding $1952+488+244+122 = 2806$
$= 0xAF6_{16}$

So, we can use 4 shifts and 3 adds instead of actual multiplication.

(5) To, reverse a bits in a register, the most efficient instruction is the grevi instruction ~~avatbtc~~ available in bit manipulation extension.
It performs the operation most efficiently.

```
grevi t1, t0, 31
```

→ In this it will reverse the bits of t0 register and it will store it in t1 register as the result.

→ This single instruction provide the whole bit reversal.

Instruction detail —

⇒ grevi rd, rs1, rmm reverses the bits of rs1 and writes the result to rd where imm is the register bits - 1

⇒ If the bit manipulation extension is not available, a multi instruction management system is needed, but grevi performs it in a single instruction.

(3) The best ~~two~~ known algorithm for swapping two registers without a temporary register uses the XOR swap trick. This works because XOR is reversible and does not need any other register other than the one being swapped.

```
xor  x5, x5, x6   # x5 = x5 ^ x6
xor  x6, x5, x6   # x6 = x5 ^ x6
                  => (x5^x6) ^ x6 = x5
xor  x5, x6, x6   # x5 = x5 ^ x6
                  => (x5^x6)^x6 = x6
```

* After these three instructions, contents of x5, and x6 are swapped.

* No additional registers are used.

* This is optimal for the RISC V processor, and follows the constraints provided in the question.

④ Given program —

```
addi  x8, x0, 0      # i=0
addi  x9, x0, 0      # sum=0
addi  x10, x0, 10    # x10=10

loop:
      beq  x8, x10, 22   # if i==10, goto 22
      add  x9, x9, x8    # sum = sum+i
      addi x8, x8, 1     # i = i+1
      j loop
22:
```

Step 1: Instruction count per loop.

The loop mainly consists of 5 instructions

→ beq
→ add
→ addi (x10)
→ addi (x11)
→ j

Step 2: Number of iterations

x10 starts from 0, increament by 10 each loop, the loop runs until x10 == x8

Given x8 = 100, (from eg), intial x10 = 0

The number of iteration is:
The number of loop = $\frac{100}{10} = 10$

But let's check the exit condition:
→ when x10 == x8(100), beq triggers and exits.
→ At that iteration, x10 has reached 100 and the loop stops

→ So for x10 = 0, 10, ... 90, 100 - when x10 is 100, beq is true and code jumps to L2 (exit)

→ This means that the beq is executed 11 times, and once for each (0, 10, 20 .... 90, 100)

Step 3: Total instructions executed

→ For the first 10 iterations (x10=0 to x10=90) all 5 instructions execute.

→ At x10=100, beq executes, and the branch is taken (jumps to L2) the branch instructions not executed.

Total instructions = 50+1 = 51

→ 10 full loops (Each width 5 instructions)
→ on the 11 th time beq runs, it successfully branches and L2 take over.

Step 4: Cycle calculation

→ Each iteration = 1 cycle (CPI = 1)
So, 51 cycles are required to complete the program.

Step 5: CPI for multicycle processor

given ideal condition

CPI = 1

CPI is defined as cycles per instructions which will remain 1 as per the information given.

(2) (a) For overall SPEC performance, geometric mean of SPEC ratios, which summarizes the performances across all benchmarks.

* opteron geometric mean SPEC ratio : 20.86
* Itanium 2 geometric mean SPEC ratio : 27.12

The higher the SPEC ratio, the better the performance.

Itanium 2 has a high geometric mean SPEC ratio (27.12), so based on overall SPEC performance, I should choose itanium 2 because it is faster on average across all benchmarks of the table.

(2)(b) Weighted average of execution time ratios = (0.6 × wupwise ratio) + (0.2 × ammp ratio) + (0.2 × apsi)

From the table —
opteron / itanium ratios:

→ wupwise = 0.92
→ ammp = 1.03
→ apsi = 0.65

Weighted average
= (0.6 × 0.92) + (0.2 × 1.03) + (0.2 × 0.65)
= 0.888

The weighted average of execution time ratios 0.888.

(2)(c) Speedup is the inverse of the execution time ratio,
Hence, Speedup = $\frac{1}{0.888}$ = 1.126

The opteron is 1.126 times faster than the itanium 2 for this workload.