COEN 210 Computer Architecture: Assignment 2

Name: Inchara Raveendra

SCU ID: 00001653600

2.5 Show how the value 0xabcdef12 would be arranged in memory of a little-endian and a big-endian machine. Assume the data are stored starting at address 0 and that the word size is 4 bytes.

Little Endian

Address	Data
12	ab
8	cd
4	ef
0	12

Big Endian

Address	Data
12	12
8	ef
4	cd
0	ab

2.6 Translate 0xabcdef12 into decimal.

a	b	c	d	e	f	1	2
10 x 16^7	11 x 16^6	12 x 16^5	13 x 16^4	14 x 16^3	15 x 16^2	1 x 16^1	2 x 16^0
2684354560	184549376	12582912	851968	57344	3840	16	2

Adding the above, 2882400018_{10}

2.7 Translate the following C code to RISC-V. Assume that the variables f, g, h, i, and j are assigned to registers x5, x6, x7, x28, and x29, respectively. Assume that the base address of the arrays A and B are in registers x10 and x11, respectively. Assume that the elements of the arrays A and B are 8-byte words:

B[8] = A[i] + A[j];

```
slli    x28, x28, 3          // x28 = i*8
ld      x28, 0(x10)          // x28 = A[i]
slli    x29, x29, 3          // x29 = j*8
ld      x29, 0(x10)          // x29 = A[j]
add  x29, x28, x29           // x29 = A[i] + A[j]
sd      x29, 64(x11)          // Store result in B[8]
```

2.8 Translate the following RISC-V code to C. Assume that the variables f, g, h, i, and j are assigned to registers x5, x6, x7, x28, and x29, respectively. Assume that the base address of the arrays A and B are in registers x10 and x11, respectively.

addi x30, x10, 8

addi x31, x10, 0

sw x31, 0(x30)

lw x30, 0(x30)

add x5, x30, x31

```
addi     x30, x10, 8                  // x30 = &A[1]

addi     x31, x10, 0                  // x31 = &A

sd     x31, 0(x30)                    // A[1] = &A

ld     x30, 0(x30)                    // x30 = A[1]

add     x5, x30, x31                  // f = A[1] + A[1] = &A + &A = 2*(&A)
```

Therefore, the translated code in C is f = 2*(&A)

2.9 For each RISC-V instruction in Exercise 2.8, show the value of the opcode (op), source register (rs1), and destination register (rd) fields. For the I-type instructions, show

the value of the immediate field, and for the R-type instructions, show the value of the second source register (rs2). For non U- and UJ-type instructions, show the funct3 field, and for R-type and S-type instructions, also show the funct7 field.

Instruction	Format	opcode	funct3	funct7	rs1	rs2	rd	imm
addi x30, x10, 8	I-type	0x13	0x0	–	10	–	x30	8
addi x31, x10, 0	I-type	0x13	0x0	–	10	–	x31	0
sd x31, 0(x30)	S-type	0x23	0x3	–	30	31	–	0
ld x30, 0(x30)	I-type	0x3	0x3	–	30	–	x30	0
add x5, x30, x31	R-type	0x33	0x0	0x0	30	31	x5	–

2.10 Assume that registers x5 and x6 hold the values 0x8000000000000000 and 0xD000000000000000, respectively.

2.10.1 What is the value of x30 for the following assembly code? add x30, x5, x6

Lets express only the highest bit of x5 and x6 in binary since the rest of the bits are zeroes.

$x30 = x5 + x6 = 1000_2 + 1101_2 = (1)\,0101_2 = 5_{10}$

Omit (1) which has overflown

Therefore x30 = 0x5000000000000000

2.10.2 Is the result in x30 the desired result, or has there been overflow?

Yes. There has been an overflow.

2.12 Provide the instruction type and assembly language instruction for the following binary value:

$0000\ 0000\ 0001\ 0000\ 1000\ 0000\ 1011\ 0011_{two}$

opcode - 0110011

rd - 00001

funct3 - 000

rs1 - 00001

rs2 - 00001

funct7 - 0000000

Referring the above values in RISC-V Reference Data Guide gives -

Instruction type: R-type

Assembly language Instruction: add x1, x1, x1

2.13 Provide the instruction type and hexadecimal representation

of the following instruction:

sw x5,32(x30)

Refer values for the below fields from RISC-V Reference Data Guide.

opcode - 0100011

rs1 - 11110

funct3 - 010

rs2 - 00101

imm - 000000100000

S-type	imm[11:5]	rs2	rs1	funct3	imm[4:0]	opcode
	0000001	00101	11110	010	00000	0100011

$0000\ 0010\ 0101\ 1111\ 0010\ 0000\ 0010\ 0011_2$

0x25F2023

2.14 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

Instruction type: R-type
0100000 00101 00111 000 00110 0110011
$0100\ 0000\ 0101\ 0011\ 1000\ 0011\ 0011\ 0011_2$

0x40538333

sub x6, x7, x5

2.35 Consider the following code:

lb x6, 0(x7)

sw x6, 8(x7)

Assume that the register x7 contains the address 0×10000000 and the data at address is 0×1122334455667788.

2.35.1 What value is stored in 0×10000008 on a big-endian machine?

0x11

2.35.2 What value is stored in 0×10000008 on a little-endian machine?

0x88

2.36 Write the RISC-V assembly code that creates the 32-bit constant $0x12345678_{hex}$ and stores that value to register x10.

0001 0010 0011 0100 0101 0110 0111 1000

lui x10, 74565 // Higher 20 bits in decimal to x10

addi x10, x10, 1656 // Add lower 12 bits in decimal to x10

2.39 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.

CPU time = Instruction Count x CPI x Clock Cycle Time

Assume Clock Cycle Time = 1 x 10^-6 s

CPU $time_{old}$ = (1x500 + 10x300 + 3x100) x 10^6 x 1 x 10^-6 = 3800s

2.39.1 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?

New Clock Cycle Time = 1.1 x 10^-6 s

CPU $time_{new}$ = (1x500x0.75 + 10x300 + 3x100) x 10^6 x 1.1 x 10^-6 = 4042.5s

Since CPU time and performance are inversely proportional, the new system with powerful arithmetic instructions has lower performance than the previous one. New system is not a good choice.

2.39.2 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?

Let Clock Cycle Time = 1 x 10^-6 s

i) When performance of arithmetic instructions doubles, $CPI_{arithmetic}$ = $CPI_{arithmetic}$/2

CPU time = (0.5x500 + 10x300 + 3x100) x 10^6 x 1 x 10^-6 = 3550s

Speed up = CPU $time_{old}$/CPU time = 3800/3550 = 1.07

The new system is 1.07 times faster

ii) When performance of arithmetic instructions improves by 10 times, $CPI_{arithmetic}$ = $CPI_{arithmetic}$/10

CPU time = (0.1x500 + 10x300 + 3x100) x 10^6 x 1 x 10^-6 = 3350s

Speed up = CPU $time_{old}$/CPU time = 3800/3350 = 1.13

The new system is 1.13 times faster

Single instruction computer (SIC) has only one instruction that can do all operations our MIPS does. The instruction has the following format

sbn a, b, c # Mem[a]=Mem[a]- Mem[b]; if (Mem[a]<0) go to PC+c

For example, here is the program to copy a number from location a to location b:

```
Start:  sbn temp, temp, 1
        sbn temp, a, 1
        sbn b,b, 1
        sbn b, temp, 1
```

Write a SIC program to add a and b, leaving the result in a and leaving b unmodified.

```
Start:  sbn temp, temp, 1  // # Mem[temp]=Mem[temp]- Mem[temp];
        sbn temp, b, 1     // # Mem[temp]=Mem[temp]- Mem[b];
        sbn a, temp, 1     // # Mem[a]=Mem[a]- Mem[temp];
```