Computer Architecture, Section 379: Homework #1

Yousef Alaa Awad

September 8, 2025

1

Given: Translate the high-level language code below into assembly instructions. The variables A, B, C, D, E and F are located in the memory and can be accessed by their label (e.g., LOAD R1, A will load A from the memory into R1). Minimize the number of instructions in the assembly code that you write.

$$F = (A - B) * \frac{(C + D)}{(E - D)}$$

A) Write the code for an accumulator architecture

$$\begin{array}{c|c}
LOAD\ E & \\
SUB\ D & E - D \\
STORE\ R1 & R1 = E - D \\
LOAD\ C & \\
ADD\ D & C + D \\
DIV\ R1 & \frac{C+D}{R1} = \frac{C+D}{E-D} \\
STORE\ R1 & R1 = \frac{C+D}{E-D} \\
LOAD\ A & \\
SUB\ B & A - B \\
MUL\ R1 & (A - B) * \frac{C+D}{E-D} \\
STORE\ F & End
\end{array}$$

All in all, this takes 11 Instructions and 11 Memory Calls.

B) Write the code for a stack architecture. Assume that the division (subtraction) operation divides (subtracts) the topmost value in the stack by the second topmost value.

$$\begin{array}{c|c}
PUSH\ D & \\
PUSH\ E & \\
SUB & E - D \\
PUSH\ C & \\
PUSH\ D & \\
ADD & C + D \\
DIV & \frac{C+D}{E-D} \\
PUSH\ B & \\
PUSH\ A & \\
SUB & A - B \\
MUL & (A - B) * \frac{C+D}{E-D} \\
POP\ F & End
\end{array}$$

All in all, this takes 12 Instructions and 7 Memory Calls.

C) Write the code for a register-memory architecture

$LOAD\ R1, C$	
$ADD\ R1, D$	
$LOAD\ R2, E$	
$SUB\ R2, D$	
$DIV\ R2, R1$	$R2 = \frac{R1}{R2} = \frac{C+D}{E-D}$
$LOAD\ R1, A$	
$SUB\ R1, B$	
$MUL\ R1, R2$	
$STORE\ R1, F$	End

All in all, this takes 9 Instructions and 7 Memory Calls.

D) Write the code for a load-store architecture

$LOAD\ R1, C$	
$LOAD\ R2, D$	
$ADD\ R1, R1, R2$	
$LOAD\ R3, E$	
$SUB\ R2, R3, R2$	$R2 = R3 - R2 = E - D$
$DIV\ R1, R1, R2$	$R1 = \frac{R1}{R2} = \frac{C+D}{E-D}$
$LOAD\ R2, A$	
$LOAD\ R3, B$	
$SUB\ R2, R2, R3$	
$MUL\ R1, R1, R2$	
$STORE\ R1, F$	End

All in all, this takes 11 Instructions and 6 Memory Calls.

E) Compare and count the number of instructions and memory accesses between the different ISAs in the previous parts of the questions (a, b, c, and d).

Type	*Instructions*	*Memory Calls*
Accumulator	11	11
Stack	12	7
Register − Memory	**9**	7
Load − Store	11	**6**

2

Given: Some architectures support the 'memory indirect' addressing mode. Below is an example. In this case, the register R2 contains a pointer to a pointer. Two memory accesses are required to load the data.

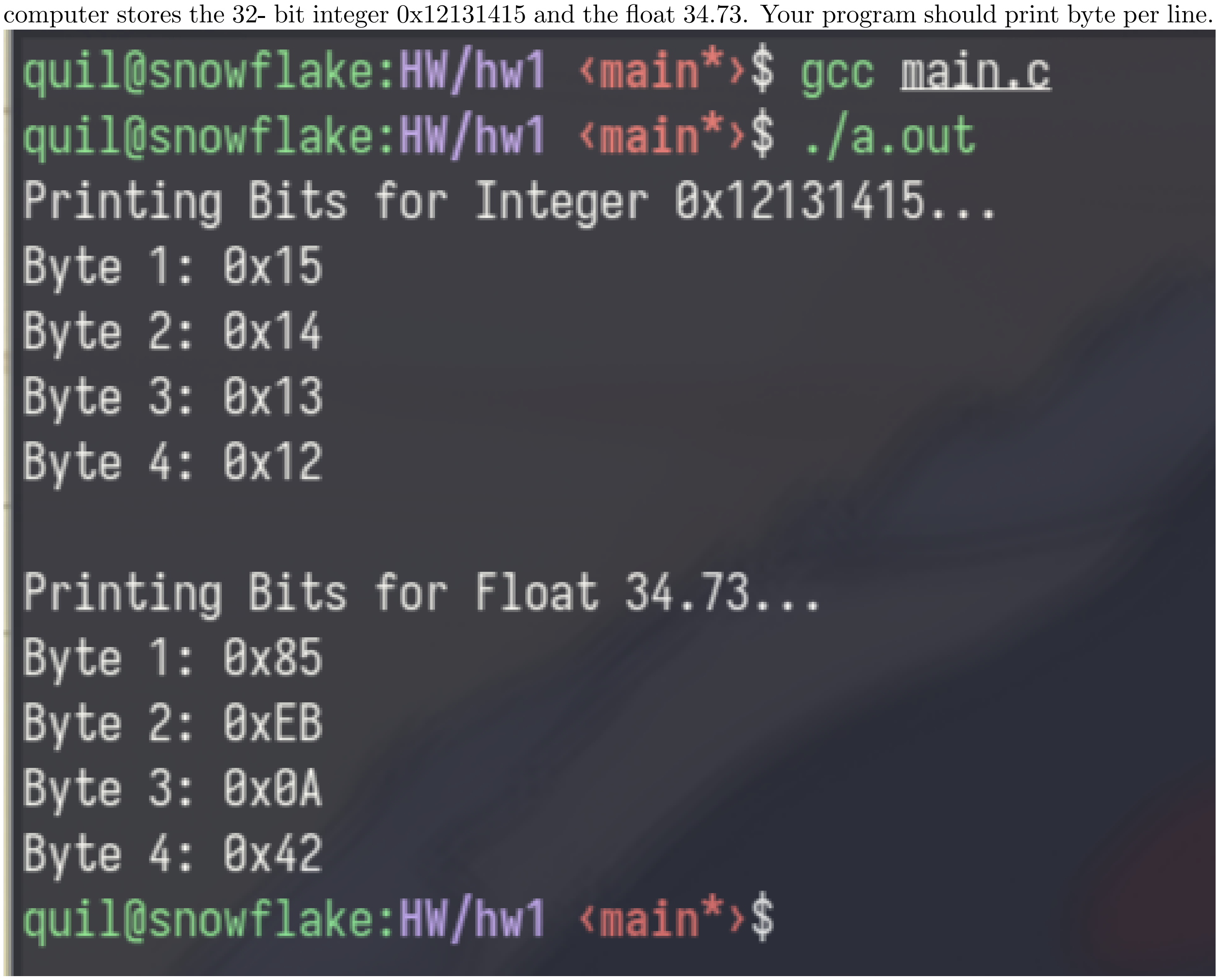
$$\text{ADD } R3, @(R2)$$

A) The MIPS CPU doesn't support this addressing mode. Write a MIPS code that's equivalent to the instruction above. The pointer-to-pointer is in register $t1. The other data is in register $t4.

$$\begin{array}{r|c} LOAD\ \$t2, (\$t1) & R1 = R2^* \\ LOAD\ \$t2, (\$t2) & R1 = R1^* \\ ADD\ \$r1, \$t4, \$t2 & End \end{array}$$

3

Given: Memory Alignment, Big Endian vs. Little Endian: Write C language program to show how your computer stores the 32- bit integer 0x12131415 and the float 34.73. Your program should print byte per line.



Listing 1: My C Code

```c
#include "stdio.h"
#include "stdint.h"

int main()
{
  // Variables
  int32_t sampleInt = 0x12131415;
  float sampleFloat = 34.73;
  // Pointer for individual finding of 2 bytes
  uint8_t *bytePointer = (uint8_t *)&sampleInt; // typecasting for funny reasons

  // Printing out the thing we are wanting
  if (printf("Printing Bits for Integer 0x12131415...\n") != 40)
  {
    return 1;
  }
  // Individually printing out 2 bytes at a time for Integer
  for (int i = 0; i < 4; i++)
  {
    if (printf("Byte %d: 0x%.2X\n", i+1, bytePointer[i]) != 13)
    {
      return 1;
    }
  }

  // Printing out the thing we are wanting x2
  if (printf("\nPrinting Bits for Float 34.73...\n") != 34)
  {
    return 1;
  }
  // Moving pointer to the float
  bytePointer = (uint8_t *)&sampleFloat;
  // Individually printing out 2 bytes at a time for Floats
  for (int i = 0; i < 4; i++)
  {
    if (printf("Byte %d: 0x%.2X \n", i+1, bytePointer[i]) != 14)
    {
      return 1;
    }
  }
  // Return success code
  return 0;
}
```