5   ISA and Microarchitecture [45 points]

You are asked to complete the following program written in MIPS assembly with a sequence of MIPS instructions that perform **64-bit integer subtraction (A - B)**. The 64-bit integer to be subtracted *from (A)* is loaded into registers $4 and $5. Similarly, the 64-bit integer to subtract (B) is loaded into registers $6 and $7. Both numbers are in two's complement form. The upper 32-bit part of each number is stored in the corresponding even-numbered register.

```
Loop:   lw  $4, 0($1)
        lw  $5, 4($1)
        lw  $6, 8($1)
        lw  $7, 12($1)

        # 64-bit subtraction
        # goes here

        addi  $1, $1, 16
        j  Loop
```

(a) [15 points] Complete the above program to perform the 64-bit subtraction explained above **using at most 4 MIPS instructions**. (*Note: A summary of the MIPS ISA is provided at the end of this question.*)

A possible sequence of instructions is as follows:

```
subu $3, $5, $7 # Subtract the least significant part
sltu $2, $5, $7 # Check if borrowing is needed
add $2, $6, $2 # Add borrow
sub $2, $4, $2 # Subtract the most significant part
```

(b) [15 points] Assume that the program executes on a pipelined processor, which does *not* implement interlocking in hardware. The pipeline assumes that all instructions are independent and relies on the compiler to properly order instructions such that there is sufficient distance between dependent instructions. The compiler either moves other independent instructions between two dependent instructions, if it can find such instructions, or otherwise, inserts nops. There is *no* internal register file forwarding (i.e., if an instruction writes into a register, another instruction *cannot* access the new value of the register until the next cycle). The pipeline does *not* implement any data forwarding. The datapath has the following *five pipeline stages*, similarly to the basic pipelined MIPS processor we discussed in lecture. Registers are accessed in the Decode stage. The execution stage contains *one ALU*.

  (a) Fetch (one clock cycle)

  (b) Decode (one clock cycle)

  (c) Execute (one clock cycle)

  (d) Memory (one clock cycle)

  (e) Write-back (one clock cycle).

Reorder the existing instructions and insert as few as possible nop instructions to correctly execute the entire program that you completed in part (a) on the given pipelined processor. Show all the instructions necessary to correctly execute the **entire program**.

We reoder the lw instructions to first load the data that corresponds to the lower parts of the two numbers since we need the lower part first. We also reorder the completely independent addi instruction to hide part of the load latency. We insert sufficient number of nop instructions until the register is written before the dependent instruction reads the same register in the decode stage.

This is the resulting code:

```
 Loop:   lw $5, 4($1)
         lw $7, 12($1)
         lw $4, 0($1)
         lw $6, 8($1)
         addi $1, $1, 16
         sltu $2, $5, $7
         subu $3, $5, $7
         nop
         nop
         add $2, $6, $2
         nop
         nop
         nop
         sub $2, $4, $2
         j Loop
```

(c) [5 points] What is the *Cycles Per Instruction (CPI)* of the program when executed on the pipelined processor provided in part (b)?

$CPI \approx 1.5$

Explanation.
Since the code is an infinite loop, the number of cycles to fill the pipeline becomes negligible after a large number of iterations. Thus, we can consider that the throughput is one instruction every cycle. We count the number of cycles for one loop iteration. It is 15 for 10 instructions. This way, $CPI \approx \frac{15}{10} = 1.5$.

(d) [10 points] Now, assume a processor with a *multi-cycle* datapath. In this multi-cycle datapath, each instruction type is executed in the following number of cycles: 4 cycles for R-type, 5 cycles for load, 4 cycles for store, and 3 cycles for jump. What is the CPI of the program in part (a) when executed on this multi-cycle datapath? Assuming the multi-cycle datapath runs at the same clock frequency as the pipelined datapath in part (b), how much speedup does pipelining provide?

CPI:

$CPI = 4.3$

Explanation.
For the multi-cycle datapath, we have to take into account the number of cycles for each instruction type: 4 cycles for R-type, 5 cycles for load, 4 cycles for store, and 3 cycles for jump.
Thus, $CPI = \frac{4 \times 5 + 5 \times 4 + 3 \times 1}{10} = 4.3$.

Speedup:

Pipelining provides 287% speedup.

Explanation.
We calculate the speedup as follows:
$Speedup = \frac{CPI_{multi-cycle}}{CPI_{pipelined}} = \frac{4.3}{1.5} = 2.87$.

MIPS Instruction Summary

Opcode	Example Assembly	Semantics
add	add $1, $2, $3	$1 = $2 + $3
sub	sub $1, $2, $3	$1 = $2 - $3
add immediate	addi $1, $2, 100	$1 = $2 + 100
add unsigned	addu $1, $2, $3	$1 = $2 + $3
subtract unsigned	subu $1, $2, $3	$1 = $2 - $3
add immediate unsigned	addiu $1, $2, 100	$1 = $2 + 100
multiply	mult $2, $3	hi, lo = $2 * $3
multiply unsigned	multu $2, $3	hi, lo = $2 * $3
divide	div $2, $3	lo = $2/$3, hi = $2 mod $3
divide unsigned	divu $2, $3	lo = $2/$3, hi = $2 mod $3
move from hi	mfhi $1	$1 = hi
move from low	mflo $1	$1 = lo
and	and $1, $2, $3	$1 = $2 & $3
or	or $1, $2, $3	$1 = $2 | $3
and immediate	andi $1, $2, 100	$1 = $2 & 100
or immediate	ori $1, $2, 100	$1 = $2 | 100
shift left logical	sll $1, $2, 10	$1 = $2 « 10
shift right logical	srl $1, $2, 10	$1 = $2 » 10
load word	lw $1, 100($2)	$1 = memory[$2 + 100]
store word	sw $1, 100($2)	memory[$2 + 100] = $1
load upper immediate	lui $1, 100	$1 = 100 « 16
branch on equal	beq $1, $2, label	if ($1 == $2) goto label
branch on not equal	bne $1, $2, label	if ($1 != $2) goto label
set on less than	slt $1, $2, $3	if ($2 < $3) $1 = 1 else $1 = 0
set on less than immediate	slti $1, $2, 100	if ($2 < 100) $1 = 1 else $1 = 0
set on less than unsigned	sltu $1, $2, $3	if ($2 < $3) $1 = 1 else $1 = 0
set on less than immediate	sltui $1, $2, 100	if ($2 < 100) $1 = 1 else $1 = 0
jump	j label	goto label
jump register	jr $31	goto $31
jump and link	jal label	$31 = PC + 4; goto label