Computer Architecture

Exercise 2.1 — Performance Improvement Calculation

You are going to enhance a machine, and there are two possible improvements: either make multiply instructions run four times faster than before, or make memory access instructions run two times faster than before. You repeatedly run a program that takes 100 seconds to execute. Of this time, 30% is used for multiplication, 45% for memory access instructions, and 25% for other tasks. What will the speedup be if you improve only multiplication? What will the speedup be if you improve only memory access? What will the speedup be if both improvements are made?

Exercise 2.2 — MIPS Register State Simulation

For each of the MIPS assembly code fragments in (a), (b), and (c), simulate execution starting in the "Start state" given below (written in hexadecimal notation) by writing down the new register state after each instruction.

Example:

Start state:

a0	0000 0190
a1	0000 03E7
t0	0123 4567
t1	89AB CDEF
s0	4E2B 3F1C
s1	251F 326D

Example Program:

```
# Start state
add  $t0, $a0, $a0
# State 1
add  $t1, $a0, $a1
# State 2
sub  $s0, $t1, $t0
# State 3
and  $s1, $s0, $a0
# State 4
```

State 1:

a0	0000 0190
a1	0000 03E7
t0	**0000 0320**
t1	89AB CDEF
s0	4E2B 3F1C
s1	251F 326D

State 2:

a0	0000 0190
a1	0000 03E7
t0	0000 0320
t1	**0000 0577**
s0	4E2B 3F1C
s1	251F 326D

State 3:

a0	0000 0190
a1	0000 03E7
t0	0000 0320
t1	0000 0577
s0	**0000 0257**
s1	251F 326D

State 4:

a0	0000 0190
a1	0000 03E7
t0	0000 0320
t1	0000 0577
s0	0000 0257
s1	**0000 0010**

(a)
```
add   $t0, $a1, $a1
addi  $t1, $t0, 0x753
sll   $s1, $a0, 2
or    $s0, $s1, $t0
and   $s0, $s0, $a1
```

(b)
```
addi  $s0, $a0, 0xE77
addi  $s1, $s0, 0xFEFC
sll   $t0, $a0, 16
or    $t0, $t0, $a1
addi  $t1, $t0, 0xABCD
```

(c)
```
lui   $t0, 0x0101
ori   $t0, $t0, 0x2323
ori   $t1, $zero, 0x4545
srl   $t1, $t1, 8
ori   $t1, $t1, 0x8989
```

(d) Write an initialisation MIPS fragment that produces the example start state without any assumptions about its own start state.

(e) Prepend your initialisation fragment to the example program and your programs of (a), (b), and (c), and single-step these through **qtspim**. Before you start, *switch the register display to hexadecimal* using the context menu in the register display pane.

Exercise 2.3

Given the following notation:

$$\begin{aligned} \mathbb{2} &:= \{0,1\} && \text{— the set of binary digits} \\ \mathbb{2}^+ &:= \text{the set of non-empty sequences of binary digits} \\ \mathbb{2}^* &:= \text{the set of sequences of binary digits} \end{aligned}$$

we define the two's complement semantics of binary digit sequences as the following function:

$$\begin{aligned} [\![\_]\!]_2 &: \mathbb{2}^+ \to \mathbb{Z} \\ [\![a_n\, a_{n-1} \ldots a_1\, a_0]\!]_2 &:= -a_n \cdot 2^n + \sum_{i=0}^{n-1} a_i \cdot 2^i \end{aligned}$$

(Here, "$a_n\, a_{n-1} \ldots a_1\, a_0$" is an $(n+1)$-element bit sequence.)

Prove the following **Sign Extension Theorem:**

For all bits $a : \mathbb{2}$ and all bit sequences $w : \mathbb{2}^*$, we have:

$$[\![a\ w]\!]_2 = [\![a\ a\ w]\!]_2$$

("$a\ w$" is the bit sequence with a as first element and w as its subsequence after the first element.)

Hint: Perform a claculation starting on one side of this equation and ending with the other side; use the method of unfolding and folding definitions.

Note: This is an extrapolation of an observation presented as an example on page 92 of the text.

Exercise 2.4 — MIPS Simulation

For each of the MIPS assembly code fragments in (a), (b), and (c), simulate execution starting in the "Start state" given below (written in hexadecimal notation) by writing down the new **register and memory state** after each instruction.

Example:

Start state:

Registers:

a0	0000 0190
a1	0000 03E7
t0	0123 4567
t1	89AB CDEF
s0	1000 0000
s1	251F 326D

Memory:

1000 0000	0000 0000
1000 0004	0000 0000
1000 0008	0123 4567
1000 000C	89AB CDEF
1000 0010	0000 0000
1000 0014	0000 0000

Example Program:

```
# Start state
sw  $a1, 0($s0)
# State 1
addi $s1, $s0, 4
# State 2
sw $a0, 0($s1)
# State 3
lw $t1, -4($s1)
# State 4
```

State 1:

a0	0000 0190
a1	0000 03E7
t0	0123 4567
t1	89AB CDEF
s0	1000 0000
s1	251F 326D

1000 0000	**0000 03E7**
1000 0004	0000 0000
1000 0008	0123 4567
1000 000C	89AB CDEF
1000 0010	0000 0000
1000 0014	0000 0000

State 2:

a0	0000 0190
a1	0000 03E7
t0	0123 4567
t1	89AB CDEF
s0	1000 0000
s1	**1000 0004**

1000 0000	0000 03E7
1000 0004	0000 0000
1000 0008	0123 4567
1000 000C	89AB CDEF
1000 0010	0000 0000
1000 0014	0000 0000

State 3:

a0	0000 0190
a1	0000 03E7
t0	0123 4567
t1	89AB CDEF
s0	1000 0000
s1	1000 0004

1000 0000	0000 03E7
1000 0004	**0000 0190**
1000 0008	0123 4567
1000 000C	89AB CDEF
1000 0010	0000 0000
1000 0014	0000 0000

State 4:

a0	0000 0190
a1	0000 03E7
t0	0123 4567
t1	**0000 03E7**
s0	1000 0000
s1	1000 0004

1000 0000	0000 03E7
1000 0004	0000 0190
1000 0008	0123 4567
1000 000C	89AB CDEF
1000 0010	0000 0000
1000 0014	0000 0000

(a)
```
lw     $t0, 0x8($s0)
lw     $t1, 0xC($s0)
add    $t0, $t0, $t1
lw     $t1, 0x10($s0)
add    $t0, $t0, $t1
```

(b)
```
addi   $s1, $s0, 0x8
lw     $t0, 0($s0)
addi   $s1, $s0, 0x4
lw     $t1, 0($s0)
add    $t0, $t0, $t1
addi   $s1, $s0, 0x4
lw     $t1, 0($s0)
add    $t0, $t0, $t1
```

(c)
```
lui    $t0, 0xFEDC
ori    $t0, 0xBA98
sw     $t0, 0($s0)
sw     $t0, 5($s0)
sw     $t0, 0xA($s0)
```

(d) Write an initialisation MIPS fragment that produces the example start state without any assumptions about its own start state.

(e) Prepend your initialisation fragment to the example program and your programs of (a), (b), and (c), and single-step these through **qtspim**. Before you start, *switch the register display to hexadecimal* using the context menu in the register display pane.