Computer Architectures
Lab 1
WinMIPS64 introduction

0) Given the following winMIPS64 processor architecture:
   - *Integer ALU: 1 clock cycle*
   - *Data memory: 1 clock cycle*
   - *Branch delay slot: 1 clock cycle*
   - Code address bus: 12
   - Data address bus: 12
   - FP multiplier unit (latency): pipelined 8 stages
   - FP arithmetic unit (latency): pipelined 6 stages
   - FP divider unit (latency): not pipelined unit, 28 clock cycles
   - Branch delay slot is disabled
   - Forwarding is enabled.

1) Write an assembly program (**`program_1.s`**) for the *MIPS64* architecture (use a text editor), able to find the maximum among 100 64-bit integer values saved in memory. The obtained value must be saved in memory using a variable called *result*.

2) Identifying the main components of the simulator:
   a. Running the *WinMIPS* simulator
      - Launch the graphic interface
      ...\winMIPS64\winmips64.exe

   b. Assembly and correct your program:
      - Load the program from the **`File→Open`** menu (*CTRL-O*). In the case the of errors, you may use the following command in the command line to compile the program and check the errors:
      `...\winMIPS64\asm program_1.s`

   c. Run your program step by step (*F7*), identifying the whole processor behavior in the six simulator windows:
      Pipeline, **Code**, **Data**, **Register**, **Cycles** and **Statistics**

   d. Disable all features present in the *Configure* menu
      a) Disable Forwarding
      b) Disable branch target buffer (*winmips64 v1.5*)
      c) Disable Delay Slot
      Execute once again your program and collect the statistics

   e. Enable one at a time the previous features (see *2.d*) menu analyzing the processor behavior, and collecting again the statistics, check the differences with respect to the ones collected in *2.d*.

3) Search in the winMIPS64 folder the following programs:

   a. `isort.s`
   b. `mult.s`
   c. `series.s`
   d. `program_1.s` (your in section 1.)

starting from the basic configuration described in the point 0), compute the time required to execute all the programs using the following configurations of the processor architecture and program weights:

1) Configuration 1
   a. Enable Forwarding
   b. Disable branch target buffer
   c. Disable Delay Slot

   Assume that the weight of all programs is the same (25%).
2) Configuration 2
   a. Enable Forwarding
   b. Enable branch target buffer
   c. Disable Delay Slot

   Assume that the weight of all programs is the same (25%).
3) Configuration 3
   a. Enable Forwarding
   b. Disable branch target buffer
   c. Enable Delay Slot

   Assume that the weight of all programs is the same (25%).
4) Configuration 4

   Configuration 1, but assume that the weight of the program `isort.s` is 50%.
5) Configuration 5

   Configuration 1, but assume that the weight of the program `mult.s` is 50%.
6) Configuration 6

   Configuration 1, but assume that the weight of the program `series.s` is 80%.

Program	Conf. 1	Conf. 2	Conf. 3	Conf. 4	Conf. 5	Conf. 6
`isort.s`						
`mult.s`						
`series.s`						
`program_1.s`						
TOTAL TIME						

4) Write an assembly program (**`program_2.s`**) for the *winMIPS64* architecture described before able to implement the following piece of code described at high-level:

```
for (i = 1; i <= 100; i++){
        v5[i] = v1[i]*v2[i];
```

```
        v6[i] = v2[i]/v3[i];
        v7[i] = v1[i]+v4[i];
}
```

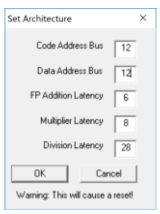
Assume that the vectors v1[], v2[], v3[], and v4[] are allocated previously in memory and contains 100 double precision floating point values; assume also that v3[] does not contain 0 values. Additionally, the vectors v5[], v6[], and v7[] are free vectors also allocated in memory.

a. Using the simulator and the configuration provided in point 0), compute how many clock cycles take the program to execute.

5) Using the WinMIPS64 simulator, validate experimentally the Amdahl's law, defined as follows:

$$\text{speedup}_{\text{overall}} = \frac{\text{execution time}_{\text{old}}}{\text{execution time}_{\text{new}}} = \frac{1}{(1-\text{fraction}_{\text{enhanced}}) + \dfrac{\text{fraction}_{\text{enhanced}}}{\text{speedup}_{\text{enhanced}}}}$$

a. Using the program developed before: **`program_2.s`**
b. Modify the processor architectural parameters related with multicycle instructions (`Menu→Configure→Architecture`) in the following way:



(a) Configuration 1
  - Change only the FP addition latency to 3

(b) Configuration 2
  - Change only the Multiplier latency to 4

(c) Configuration 1
  - Change only the division latency to 12

Compare the results obtained by simulation in the three different configurations against the ones calculated by hand using the Amdahl's law in every case.

Appendix: *winMIPS64 Instruction Set*

WinMIPS64

<u>The following assembler directives are supported</u>

.data - start of data segment
.text - start of code segment
.code - start of code segment (same as .text)
.org <n> - start address
.space <n> - leave n empty bytes
.asciiz <s> - enters zero terminated ascii string
.ascii <s> - enter ascii string
.align <n> - align to n-byte boundary
.word <n1>,<n2>.. - enters word(s) of data (64-bits)
.byte <n1>,<n2>.. - enter bytes
.word32 <n1>,<n2>.. - enters 32 bit number(s)
.word16 <n1>,<n2>.. - enters 16 bit number(s)
.double <n1>,<n2>.. - enters floating-point number(s)

where <n> denotes a number like 24, <s> denotes a string like "fred", and
<n1>,<n2>.. denotes numbers seperated by commas.

<u>The following instructions are supported</u>

lb - load byte
lbu - load byte unsigned
sb - store byte
lh - load 16-bit half-word
lhu - load 16-bit half word unsigned
sh - store 16-bit half-word
lw - load 32-bit word
lwu - load 32-bit word unsigned
sw - store 32-bit word
ld - load 64-bit double-word
sd - store 64-bit double-word
l.d - load 64-bit floating-point
s.d - store 64-bit floating-point
halt - stops the program

daddi - add immediate
daddui - add immediate unsigned
andi - logical and immediate
ori - logical or immediate
xori - exclusive or immediate
lui - load upper half of register immediate
slti - set if less than or equal immediate
sltiu - set if less than or equal immediate unsigned

beq - branch if pair of registers are equal
bne - branch if pair of registers are not equal
beqz - branch if register is equal to zero
bnez - branch if register is not equal to zero

j - jump to address
jr - jump to address in register
jal - jump and link to address (call subroutine)
jalr - jump and link to address in register (call subroutine)

dsll - shift left logical
dsrl - shift right logical
dsra - shift right arithmetic
dsllv - shift left logical by variable amount
dsrlv - shift right logical by variable amount
dsrav - shift right arithmetic by variable amount
movz - move if register equals zero
movn - move if register not equal to zero
nop - no operation
and - logical and
or - logical or
xor - logical xor
slt - set if less than
sltu - set if less than unsigned
dadd - add integers
daddu - add integers unsigned
dsub - subtract integers
dsubu - subtract integers unsigned

add.d - add floating-point
sub.d - subtract floating-point
mul.d - multiply floating-point
div.d - divide floating-point
mov.d - move floating-point
cvt.d.l - convert 64-bit integer to a double FP format
cvt.l.d - convert double FP to a 64-bit integer format
c.lt.d - set FP flag if less than
c.le.d - set FP flag if less than or equal to
c.eq.d - set FP flag if equal to
bc1f - branch to address if FP flag is FALSE
bc1t - branch to address if FP flag is TRUE
mtc1 - move data from integer register to FP register
mfc1 - move data from FP register to integer register