# Lab 6: TOMASULO'S ALGORITHM : EXECUTION OF PROGRAMS

Computer Architecture and Engineering (3rd year)
E.T.S. de Ingeniería Informática (ETSINF)
Dpto. de Informática de Sistemas y Computadores (DISCA)

## Goals:

- Analyze the influence of the design parameters of the superscalar processor on programs' performance.

## Development:

For the development of the lab, we will start with a RISC-V superscalar simulator, which incorporates out-of-order execution and speculation based on the Tomasulo's algorithm. The simulator accepts a reduced set of integer instructions, and double precision floating point arithmetic and load/store instructions.

The following syntax will be used to execute the simulator:

```
./riscv-ooo [OPTIONS] -f <file.s>

----- Execution -----

-j
        It generates a single .htm file with all the results.
-s
        No .htm files are generated during execution.

----- Configuration -----

        NOTE: A value of 0 in any parameter indicates that the default value is u

-i <num>:<lat>:<segm>
        Number and latency of integer/jump operators,
        and type ([s] for pipelined/[c] for conventional)
-a <num>:<lat>:<segm>
        Number and latency of addition/substraction/comparison operators,
        and type ([s] for pipelined/[c] for conventional)
-m <num>:<lat>:<segm>
        Number and latency of multiplication/division operators,
        and type ([s] for pipelined/[c] for conventional)
-l <num>:<lat>:<segm>
        Number and latency of memory operators,
        and type ([s] for pipelined/[c] for conventional)
-e <es>:<sr>:<md>:<bl>:<be>
```

```
        Number of reservation stations for integers/branches,
        addition/substraction/comparison, multiplication/division
        and read and write buffers
-r <num>
        Reorder Buffer entries
-v <issue>:<buses>:<commits>
        Number of ways of superscalar processor in ISSUE, BUSES and COMMIT
-M
        Number of operators and reservation stations depending on the number
        of ways.
-p {1|2h|2s|p|c}
        Predictor type.
           [1] : 1 bit (default)
           [2h]: 2 bits with hysteresis
           [2s]: 2 bits with saturation
           [p] : Perfect predictior
           [c] : BTB with perfect predictor
           [pnt] : Predict-not-taken
-b <num>
        Number of BTB entries

----- Input -----

-f <file.s>
        Name of the assembly input file
```

## Implemented Instructions

| Integer | Floating point |
|---|---|
| LD Rx, disp(Ry) | FLD Fx, disp(Ry) |
| SD Ry, disp(Rx) | FSD Fy, disp(Rx) |
| ADD Rx, Ry, Rz | FADD.D Fx, Fy, Fz |
| SUB Rx, Ry, Rz | FSUB.D Fx, Fy, Fz |
| ADDI Rx, Ry, value | |
| SUBI Rx, Ry, value | |
| | FMUL.D Fx, Fy, Fz |
| | FDIV.D Fx, Fy, Fz |
| | FLT.D Fx, Fy, Fz |
| BEQZ Rx, disp | |
| BNEZ Rx, disp | |
| ECALL | |

## Exercices

1. Influence of the number of ways in execution results.

   Execute the simulator by modifying the number of ways of the processor. To avoid
   the influence of the rest of design parameters on results, configure them to have

enough ressources. To do so, execute the simulator using option "-M", which introduce operators and reservation stations attending to the number of ways in the superscalar processor, and a *reorder buffer* with 128 entries.

Use the following test programs:

- `daxpy64.s`. Implements the operation $\vec{Z} = a \cdot \vec{X} + \vec{Y}$ (DAXPY loop) using vectors of 64 floating point components.
- `matmul.s`. Multiplies two matrices $A$ and $B$ of $5 \times 5$ floating point numbers.

For each case, analyze the processor performance under different control hazard strategies: *predict-not-taken*, 1-bit BTB predictor, 2-bit BTB predictor with saturation and a perfect predictor. The perfect predictor is simulated by executing programs two consecutive times. During the first execution, results from program branch instructions are stored in an auxiliary file. During the second execution, such auxiliary file is used for prediction, which will result in a 100 % of hit prediction.

## **daxpy64.s**

a) Pipelined processor. Execute the following commands:

```
./riscv-ooo -f daxpy64.s -s -r 128 -M -p pnt > daxpy64-v1-pnt
./riscv-ooo -f daxpy64.s -s -r 128 -M -p 1   > daxpy64-v1-p1
./riscv-ooo -f daxpy64.s -s -r 128 -M -p 2s  > daxpy64-v1-p2s
./riscv-ooo -f daxpy64.s -s -r 128 -M -p p   > daxpy64-v1-pp
```

b) 2-way superscalar processor. Execute the following commands:

```
./riscv-ooo -f daxpy64.s -s -r 128 -v 2:2:2 -M -p pnt > daxpy64-v2-pnt
./riscv-ooo -f daxpy64.s -s -r 128 -v 2:2:2 -M -p 1   > daxpy64-v2-p1
./riscv-ooo -f daxpy64.s -s -r 128 -v 2:2:2 -M -p 2s  > daxpy64-v2-p2s
./riscv-ooo -f daxpy64.s -s -r 128 -v 2:2:2 -M -p p   > daxpy64-v2-pp
```

c) 4-way superscalar processor. Execute the following commands:

```
./riscv-ooo -f daxpy64.s -s -r 128 -v 4:4:4 -M -p pnt > daxpy64-v4-pnt
./riscv-ooo -f daxpy64.s -s -r 128 -v 4:4:4 -M -p 1   > daxpy64-v4-p1
./riscv-ooo -f daxpy64.s -s -r 128 -v 4:4:4 -M -p 2s  > daxpy64-v4-p2s
./riscv-ooo -f daxpy64.s -s -r 128 -v 4:4:4 -M -p p   > daxpy64-v4-pp
```

d) 8-way superscalar processor. Execute the following commands:

```
./riscv-ooo -f daxpy64.s -s -r 128 -v 8:8:8 -M -p pnt > daxpy64-v8-pnt
./riscv-ooo -f daxpy64.s -s -r 128 -v 8:8:8 -M -p 1   > daxpy64-v8-p1
./riscv-ooo -f daxpy64.s -s -r 128 -v 8:8:8 -M -p 2s  > daxpy64-v8-p2s
./riscv-ooo -f daxpy64.s -s -r 128 -v 8:8:8 -M -p p   > daxpy64-v8-pp
```

## `matmul.s`

*a*) Pipelined processor. Execute the following commands:

```
./riscv-ooo -f matmul.s -s -r 128 -M -p pnt     > matmul-v1-pnt
./riscv-ooo -f matmul.s -s -r 128 -M -p 1 -b 3  > matmul-v1-p1
./riscv-ooo -f matmul.s -s -r 128 -M -p 2s -b 3 > matmul-v1-p2s
./riscv-ooo -f matmul.s -s -r 128 -M -p p       > matmul-v1-pp
```

*b*) 2-way superscalar processor. Execute the following commands:

```
./riscv-ooo -f matmul.s -s -r 128 -v 2:2:2 -M -p pnt     > matmul-v2-pnt
./riscv-ooo -f matmul.s -s -r 128 -v 2:2:2 -M -p 1 -b 3  > matmul-v2-p1
./riscv-ooo -f matmul.s -s -r 128 -v 2:2:2 -M -p 2s -b 3 > matmul-v2-p2s
./riscv-ooo -f matmul.s -s -r 128 -v 2:2:2 -M -p p       > matmul-v2-pp
```

*c*) 4-way superscalar processor. Execute the following commands:

```
./riscv-ooo -f matmul.s -s -r 128 -v 4:4:4 -M -p pnt     > matmul-v4-pnt
./riscv-ooo -f matmul.s -s -r 128 -v 4:4:4 -M -p 1 -b 3  > matmul-v4-p1
./riscv-ooo -f matmul.s -s -r 128 -v 4:4:4 -M -p 2s -b 3 > matmul-v4-p2s
./riscv-ooo -f matmul.s -s -r 128 -v 4:4:4 -M -p p       > matmul-v4-pp
```

*d*) 8-way superscalar processor. Execute the following commands:

```
./riscv-ooo -f matmul.s -s -r 128 -v 8:8:8 -M -p pnt     > matmul-v8-pnt
./riscv-ooo -f matmul.s -s -r 128 -v 8:8:8 -M -p 1 -b 3  > matmul-v8-p1
./riscv-ooo -f matmul.s -s -r 128 -v 8:8:8 -M -p 2s -b 3 > matmul-v8-p2s
./riscv-ooo -f matmul.s -s -r 128 -v 8:8:8 -M -p p       > matmul-v8-pp
```

$\Rightarrow$ Analyze the execution time and the IPC (inverse of CPI). Fill the following tables:

Program: `daxpy64.s`:

| T exec (cycles) | -p pnt | -p 1 | -p 2s | -p p | IPC | -p pnt | -p 1 | -p 2s | -p p |
|---|---|---|---|---|---|---|---|---|---|
| 1 way | 1608 | 622 | 622 | 601 | 1 way | 0.36 | 0.94 | 0.94 | 0.97 |
| 2 ways | 1411 | 361 | 361 | 340 | 2 ways | 0.41 | 1.61 | 1.61 | 1.71 |
| 4 ways | 1347 | 235 | 235 | 212 | 4 ways | 0.43 | 2.48 | 2.48 | 2.75 |
| 8 ways | 1284 | 172 | 172 | 150 | 8 ways | 0.45 | 3.39 | 3.39 | 3.89 |

Program: `matmul.s`:

| T exec (cycles) | -p pnt | -p 1 | -p 2s | -p p | IPC | -p pnt | -p 1 | -p 2s | -p p |
|---|---|---|---|---|---|---|---|---|---|
| 1 way | 5774 | 4183 | 3611 | 2838 | 1 way | 0.36 | 0.5 | 0.58 | 0.74 |
| 2 ways | 4791 | 2905 | 2265 | 1434 | 2 ways | 0.44 | 0.72 | 0.93 | 1.46 |
| 4 ways | 4457 | 2291 | 1571 | 734 | 4 ways | 0.47 | 0.92 | 1.34 | 2.86 |
| 8 ways | 4408 | 2266 | 1522 | 568 | 8 ways | 0.48 | 0.93 | 1.38 | 3.70 |

$\Rightarrow$ Plot in a graph these results and interpret them. As an advice, plot in the y-axis the performance (execution time or IPC), in the x-axis the number of ways, and use the type of predictor as parameter. Answer the following questions:

- Which is the influence of the number of ways in performance?

If we increase the number of ways, the overall execution time decreases and the IPC improves

From best to worst: perfect prediction (p), 2-bit with saturation, 1-bit predictor and finally predict-not-taken

- Which is the influence of the type of predictor in performance?
- Does it exist any relation between the number of ways, the type of predictor and the exhibited performance?

It doesn't seem to

2. Influence of the size of the *reorder buffer* in performance.

Using the program `matmul.s`, execute the simulator with different values for the size of the *reorder buffer*. Use a 2-way and 4-way superescalar processor and measure performance using a 1-bit and a perfect predictor.

Following the same approach, configure the rest of design parameters to have enough resources for the program execution (use option "-M" ).

Execute the simulations using a different number of ways and *reorder buffer* size:

```
./riscv-ooo -f matmul.s -s -r 128 -v 2:2:2 -M -p 1  > matmul-v2-r128-p1
./riscv-ooo -f matmul.s -s -r 64 -v 2:2:2 -M -p 1  > matmul-v2-r64-p1
./riscv-ooo -f matmul.s -s -r 32 -v 2:2:2 -M -p 1  > matmul-v2-r32-p1
./riscv-ooo -f matmul.s -s -r 16 -v 2:2:2 -M -p 1  > matmul-v2-r16-p1
./riscv-ooo -f matmul.s -s -r 8 -v 2:2:2 -M -p 1  > matmul-v2-r8-p1
./riscv-ooo -f matmul.s -s -r 4 -v 2:2:2 -M -p 1  > matmul-v2-r4-p1

./riscv-ooo -f matmul.s -s -r 128 -v 2:2:2 -M -p p  > matmul-v2-r128-pp
./riscv-ooo -f matmul.s -s -r 64 -v 2:2:2 -M -p p  > matmul-v2-r64-pp
./riscv-ooo -f matmul.s -s -r 32 -v 2:2:2 -M -p p  > matmul-v2-r32-pp
./riscv-ooo -f matmul.s -s -r 16 -v 2:2:2 -M -p p  > matmul-v2-r16-pp
./riscv-ooo -f matmul.s -s -r 8 -v 2:2:2 -M -p p  > matmul-v2-r8-pp
./riscv-ooo -f matmul.s -s -r 4 -v 2:2:2 -M -p p  > matmul-v2-r4-pp

./riscv-ooo -f matmul.s -s -r 128 -v 4:4:4 -M -p 1  > matmul-v4-r128-p1
./riscv-ooo -f matmul.s -s -r 64 -v 4:4:4 -M -p 1  > matmul-v4-r64-p1
./riscv-ooo -f matmul.s -s -r 32 -v 4:4:4 -M -p 1  > matmul-v4-r32-p1
./riscv-ooo -f matmul.s -s -r 16 -v 4:4:4 -M -p 1  > matmul-v4-r16-p1
./riscv-ooo -f matmul.s -s -r 8 -v 4:4:4 -M -p 1  > matmul-v4-r8-p1
./riscv-ooo -f matmul.s -s -r 4 -v 4:4:4 -M -p 1  > matmul-v4-r4-p1

./riscv-ooo -f matmul.s -s -r 128 -v 4:4:4 -M -p p  > matmul-v4-r128-pp
./riscv-ooo -f matmul.s -s -r 64 -v 4:4:4 -M -p p  > matmul-v4-r64-pp
./riscv-ooo -f matmul.s -s -r 32 -v 4:4:4 -M -p p  > matmul-v4-r32-pp
./riscv-ooo -f matmul.s -s -r 16 -v 4:4:4 -M -p p  > matmul-v4-r16-pp
./riscv-ooo -f matmul.s -s -r 8 -v 4:4:4 -M -p p  > matmul-v4-r8-pp
./riscv-ooo -f matmul.s -s -r 4 -v 4:4:4 -M -p p  > matmul-v4-r4-pp
```

⇒ Analyze the resulting execution time and IPC. Fill the following tables:

Program: `matmul.s`:

| T ejec (ciclos) Size ROB | | -p 1 | -p p | | IPC | | -p 1 | -p p |
|---|---|---|---|---|---|---|---|---|
| 2 ways | -r 128 | 2916 | 1434 | | 2 ways | -r 128 | 0,72 | 1,46 |
| | -r 64 | 2916 | 1434 | | | -r 64 | 0,72 | 1,46 |
| | -r 32 | 3171 | 1898 | | | -r 32 | 0,66 | 1,11 |
| | -r 16 | 3446 | 2757 | | | -r 16 | 0,61 | 0,76 |
| | -r 8 | 4671 | 4241 | | | -r 8 | 0,45 | 0,49 |
| | -r 4 | 5948 | 5727 | | | -r 4 | 0,35 | 0,37 |
| | | | | | | | | |
| 4 ways | -r 128 | 2301 | 734 | | 4 ways | -r 128 | 0,91 | 2,86 |
| | -r 64 | 2301 | 940 | | | -r 64 | 0,91 | 2,23 |
| | -r 32 | 2801 | 1691 | | | -r 32 | 0,75 | 1,24 |
| | -r 16 | 3251 | 2565 | | | -r 16 | 0,65 | 0,82 |
| | -r 8 | 4579 | 4239 | | | -r 8 | 0,46 | 0,50 |
| | -r 4 | 5882 | 5706 | | | -r 4 | 0,36 | 0,37 |

⇒ Plot results in a graph and interpret them. As an advice, show in the y-axis the performance (execution time or IPC), in the x-axis the size of the ROB and use as parameter the combination of number of ways and type of predictor. Answer the following questions:

- How does the ROB size impact performance? The larger the ROB, the less cycles the program lasts

- Is there any relation between the number of ways, the ROB size and the exhibited performance? Yes: the higher the number of ways and ROB size, the lower the program's duration is in cycles, which implies that the program has greater performance

- Is there any relation between the type of predictor, the ROB size and the exhibited performance? Yes: when we increment the ROB, if we are using a perfect predictor, the performance is higher, as we perform more instructions per cycle

3. Dimensioning the resources in the processor execution unit.

Using the simulator, analyze the degree of usage of processor resources. This analysys, carried out on a large sample of test programs, could be of interest to properly dimension the processor resource.

Use the program `daxpy64.s`,with 4-ways and a ROB size of 128. In the simulator results, the section "Ocupación de recursos" provides the maximum degree of usage of reservation stations, operators and ROB. The following table shows the result for a 1-bit predictor, available at file `daxpy64-v4-p1`.

| | -p 1 | -p 1 |
|---|---|---|
| Integer RS | 5 | |
| FP A/S RS | 6 | |
| FP M/D RS | 4 | -e 5:6:4:4:7 |
| Load Buf. | 4 | |
| Store Buf. | 7 | |
| Integer | 4 | -i  4:1:c |
| FP A/S Op. | 2 | -a  2:4:c |
| FP M/D Op. | 3 | -m 3:7:c |
| Mem(AC) Op. | 3 | |
| Mem Op. | 3 | -l  3:2:c |
| RoB entries | 51 | -r 51 |

-f: file       -s: do not generate hmtl while execution     -v: number of ways in ISSUE:BUSES:COMMIT     -l: num and latency of mem
-i:number and latency of int and branch     -a: num and latency of arithmetic op   -m: num and latency of multi/div
-e: num of reservation stations for [INT/Branch : ARITHM : MUL/DIV : Load : Stores]
-p: predictor type                     -r: ROB buffers

AIC Labs 3rd. year, ETSINF - UPV                                          Lab. 6

It can be check that a configuration with such number of resources, instead of using "-M", will be enough to obtain the same performance:

```
./riscv-ooo -f daxpy64.s -s -v 4:4:4 -i 4:1:c -a 2:4:c -m 3:7:c \
   -l 3:2:c -e 5:6:4:4:7 -p 1 -r 51
```

Once the usage of resources is known, tests can be carried out to analyze the impact on performance of resources reduction. For instance:

- Reduce the number of multiplication units from 3 to 2:

  ```
  ./riscv-ooo -f daxpy64.s -s -v 4:4:4 -i 4:1:c -a 2:4:c -m 2:7:c \
     -l 3:2:c -e 5:6:4:4:7 -p 1 -r 51
  ```

- Use a pipelined multiplier:

  ```
  ./riscv-ooo -f daxpy64.s -s -v 4:4:4 -i 4:1:c -a 2:4:c -m 1:7:s \
     -l 3:2:c -e 5:6:4:4:7 -p 1 -r 51
  ```

- Use a pipelined multiplier and reduce the integer operators from 4 to 2:

  ```
  ./riscv-ooo -f daxpy64.s -s -v 4:4:4 -i 2:1:c -a 2:4:c -m 1:7:s \
     -l 3:2:c -e 5:6:4:4:7 -p 1 -r 51
  ```

- Use a pipelined multiplier, reduce the number of integer operators from 4 to 2 and the number of memory operators from 3 to 2:

  ```
  ./riscv-ooo -f daxpy64.s -s -v 4:4:4 -i 2:1:c -a 2:4:c -m 1:7:s \
     -l 2:2:c -e 5:6:4:4:7 -p 1 -r 51
  ```

⇒ Analyze the usage of resources for two control hazard strategies: predict-not-taken and perfect prediction. Results are already available in files `daxpy64-v4-pnt` and `daxpy64-v4-pp`. Fill the following table:

|  | -p pnt | -p p |
|---|---|---|
| Integer RS | 5 | 6 |
| FP A/S RS | 1 | 6 |
| FP M/D RS | 1 | 5 |
| Load Buf. | 3 | 5 |
| Store Buf. | 1 | 7 |
| Integer | 4 | 4 |
| FP A/S Op. | 1 | 2 |
| FP M/D Op. | 1 | 3 |
| Mem(AC) Op. | 3 | 3 |
| Mem Op. | 3 | 3 |
| RoB entries | 12 | 54 |

⇒ Analyze the impact of reducing available resources on performance following the same approach previously defined for a 1-bit predictor.