Classifying Superscalar Processors

Bruno G. Vergara, João Pedro F. Pereira, Pedro C. Beck
Institute of Informatics, Federal University of Rio Grande do Sul - UFRGS - Porto Alegre, Brazil
00324256 bgvergara@inf.ufrgs.br, 00324521 jpfpereira@inf.ufrgs.br, 00324055 pcbeck@inf.ufrgs.br

***Abstract*—In this article we will discuss whether processors with a superscalar pipeline can be classified, according to Flynn's taxonomy, as SISD, SIMD, or MIMD.**

We will also present our processors, show the SIMD extensions available on each one of them; and compare how a C program is compiled with and without AVX support using the GNU Compiler Collection (GCC).

***Index Terms*—superscalar pipeline, Flynn's taxonomy**

I. Scalar Processors

Scalar processors are a class of processors that have an instruction pipeline to implement instruction-level parallelism (ILP). These processors are able to process multiple instructions in parallel, depending on the size of the pipeline, with each instruction starting right after the previous instruction gets into another cycle, with an overlap in their execution.

A. Instruction Pipeline

Instruction pipelines are used in scalar processors to be able to execute more instructions per cycle.

On the MIPS architecture each instruction takes at most 5 cycles to complete, with each clock cycle being: *instruction fetch cycle* (IF), *instruction decode/register fetch cycle* (ID), *execution/effective address cycle* (EX), *memory access* (MEM), and *write-back cycle* (WB). The pipeline works such that whenever the first instruction finishes the IF cycle, the second instruction starts its IF cycle; then in the second clock cycle the first two instructions will be executed, the first in the ID cycle and the second in the IF cycle. Then with every next cycle one instruction will begin its execution and after 5 cycles there will be 5 instructions being executed in parallel, as illustrated in Fig. 1.

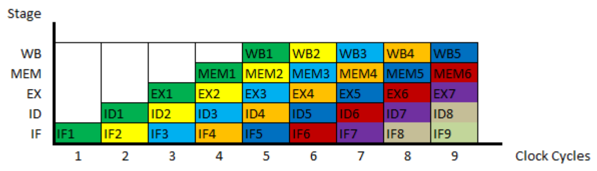

Stage
WB
MEM
EX
ID
IF
WB1 WB2 WB3 WB4 WB5
MEM1 MEM2 MEM3 MEM4 MEM5 MEM6
EX1 EX2 EX3 EX4 EX5 EX6 EX7
ID1 ID2 ID3 ID4 ID5 ID6 ID7 ID8
IF1 IF2 IF3 IF4 IF5 IF6 IF7 IF8 IF9
1 2 3 4 5 6 7 8 9
Clock Cycles

Fig. 1. Example of a 5 stage pipeline.

II. Flynn's Taxonomy

The Flynn's Taxonomy is a classification of computer architectures proposed by Michael J. Flynn, in 1966. It divides computer architectures in four different categories, using the stream concept to categorize the architectures. The concepts used to create these categories are instruction streams and data streams. These concepts originate four different classes of computer architectures: Single Instruction Stream - Single Data Stream (SISD), Single Instruction Stream – Multiple Data Streams (SIMD), Multiple Instruction Streams – Single Data Streams (MISD) and Multiple Instruction Streams – Multiple Data Streams (MIMD).

A. Single Instruction stream, Single Data stream

The SISD category corresponds to uniprocessor machines devoid of parallelism in both the data and the instruction streams. Therefore, SISD machines can only work in a sequential manner, executing one instruction and operating one data stream at a time.

B. Single Instruction stream, Multiple Data streams

The SIMD category represents multiprocessor machines able to operate multiple data streams while executing only one instruction stream. Thus, SIMD architecture enables the performance of operations, such as those which involve vectors and matrix operators, in a parallel way.

C. Multiple Instruction streams, Single Data stream

MISD is a theoretical category with no real-world examples. A MISD processor would have multiple instruction streams operating in a single data stream. In practice, such processors would have to simultaneously write multiple times over the same data, making it hard to do anything meaningful with it.

D. Multiple Instruction streams, Multiple Data streams

MIMD architectures are designed to allow operations in several data and instruction streams, which are accessed through multiple cores or threads in a single processor. Differently from the other categories, MIMD processors operate in asynchronous time.

III. Superscalar Processors

Superscalar processors are a type of scalar processors that are able to simultaneously fetch, decode and execute more than one instruction per clock cycle. These processors use techniques such as parallel instruction decoding, parallel register renaming, speculative execution, and out-of-order execution.

A superscalar processor with dual 5 stage pipelines as the one seen in section I.A. will be able to process 2 instructions in each clock cycle in each of the 5 stages, and after 5 cycles there will be 10 instructions being executed in parallel, as illustrated in Fig. 2.

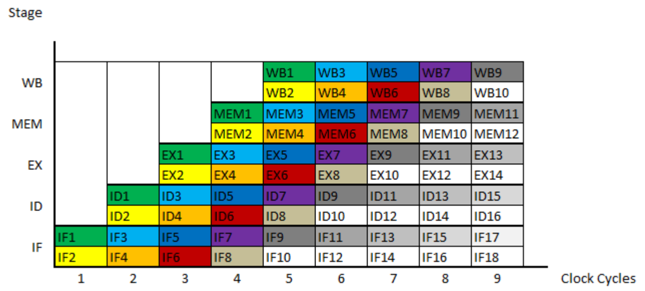

Stage
WB
MEM
EX
ID
IF
WB1 WB3 WB5 WB7 WB9
WB2 WB4 WB6 WB8 WB10
MEM1 MEM3 MEM5 MEM7 MEM9 MEM11
MEM2 MEM4 MEM6 MEM8 MEM10 MEM12
EX1 EX3 EX5 EX7 EX9 EX11 EX13
EX2 EX4 EX6 EX8 EX10 EX12 EX14
ID1 ID3 ID5 ID7 ID9 ID11 ID13 ID15
ID2 ID4 ID6 ID8 ID10 ID12 ID14 ID16
IF1 IF3 IF5 IF7 IF9 IF11 IF13 IF15 IF17
IF2 IF4 IF6 IF8 IF10 IF12 IF14 IF16 IF18
1 2 3 4 5 6 7 8 9
Clock Cycles

Fig. 2. Example of a superscalar 5 stage pipeline.

IV. Classification of Superscalar Processors

There is an ongoing debate on the classification of superscalar processors according to Flynn's Taxonomy. Depending on the characteristics of each superscalar processor, their classification may be different. Since MISD is a theoretical category, pipelined superscalar processors could be placed in one of the three other categories: SISD, SIMD or MIMD.

Some examples of superscalar processors and their classifications according to Flynn's Taxonomy are:

A. Motorola 68060

Motorola 68060 was a superscalar processor from the Motorola 68000 family, featuring a dual four-stage pipeline to decode, fetch and execute instructions. This processor had a single core with no multithreading capabilities, hence a single instruction stream, and it had no SIMD instructions extensions.

Since it has a single instruction stream and no support for multiple data instructions, the Motorola 68060 can be classified as a superscalar SISD processor.

B. Intel Pentium MMX

The Pentium MMX was a superscalar processor from Intel. It featured a superscalar architecture with two instruction pipelines, and with the introduction of the MMX instruction set, a SIMD extenstion to the original Intel Pentium, making it possible to process multiple streams of data, this processor was capable of executing SIMD instructions.

Since the Pentium MMX had a single core and lacked any type of multithreading abilities, it had a single instruction stream; but since it had instructions that could process multiple streams of data, this processor could be considered a superscalar SIMD processor.

C. Intel Core i7

The Intel Core i7 line of processors are multicore superscalar processors. The desktop versions have four to eight CPU cores and feature hyper-threading, allowing them to have multiple streams of instructions and data occurring at the same time. Since Sandy Bridge, they also support vector processing instructions, capable of performing a single instruction on multiple data streams.

Since there are multiple streams of instructions able to manipulate multiple streams of data, they can be considered superscalar MIMD processors.

V. Conclusion

As seen in the examples before, despite the fact that they are all superscalar processors, their classification according to Flynn's Taxonomy varies. The categorization of an architecture depends more on their other characteristics than on the fact of it being superscalar or not. We have examples of SISD, SIMD and MIMD superscalar architectures. The most important factors to classify a processor are its capability to manipulate vectors and the existence of multicore or multithreading capabilites.

There is no unique answer to the problem of classifying pipelined superscalar processors. The existence of a superscalar pipeline doesn't alter the quantity of instruction streams originating from a processor, or the quantity of data streams this processor is operating in.

Our Processors

A. Intel Core i7-8700K

The Intel Core i7-8700K is a 64-bit microprocessor from the 8th generation of the Intel Core family. It has 6 cores and 12 threads, and uses the x86 instruction set, that has many SIMD extensions. Some examples of these extensions are:

1) AES-NI: According to [11] in page 61, the Intel Advanced Encryption Standard New Instructions (Intel AES-NI) are a set of SIMD instructions that are responsible for implementing fast and secure encryption and decryption of data based on the Advanced Encryption Standard (AES). Those instructions are used for a wide range of applications concerning cryptography, such as operations of encryption/decryption, authentication, random number generation, and authenticated encryption.

2) AVX2: As stated in page 65 of [11], the Intel Advanced Vector Extensions 2.0 (Intel AVX2) implements 256-bit integer instructions, floating-point fused multiply add (FMA) instructions, gather operations, and serves as a expansion for the Intel Advanced Vector Extensions (Intel AVX) - which are designed to achieve higher throughput to certain integer and floating point operation.

3) SSE4.2: According to [11] in page 12, this processor has the SSE4.2 instruction set extension. Streaming SIMD Extensions 4.2 (SSE4.2) is a set of SIMD instructions designed to, according to [12], improve the performance of various applications, such as video encoders, image processing, 3D games, and string/text processing.

B. IBM PowerPC 970FX

The PowerPC 970FX, also known as PowerPC G5, is a 64-bit RISC microprocessor from IBM. It uses the POWER instruction set developed by IBM. Due to having a RISC architecture, it doesn't feature many instruction extensions, but it has SIMD capabilities nevertheless.

The PowerPC 970FX has a Vector Processing Unit (VPU), as seen in page 345 of [10], which is an extension to the PowerPC architecture that adds a 128-bit vector execution unit.

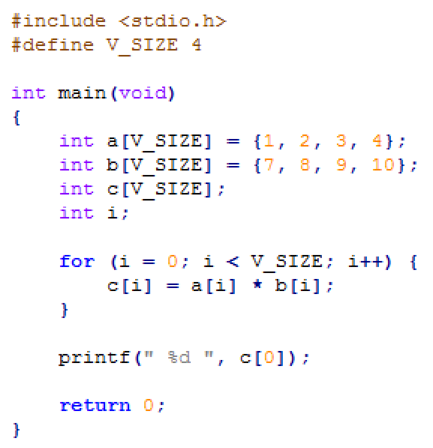
This unit allows the simultaneous execution of up to four 32-bit floating operations or sixteen 8-bit fixed-point operations in one instruction.

Compiling with and without AVX instructions

Advanced Vector Extensions, also known as AVX, is a set of instructions available on some Intel processors to optimize operations with vectors.

To show an example of AVX instructions in use, we wrote a simple program that manipulates vectors (Fig. 3.) and compiled with the GNU Compiler Collection (GCC) with and without AVX instructions, using the following commands to extract the assembly code:

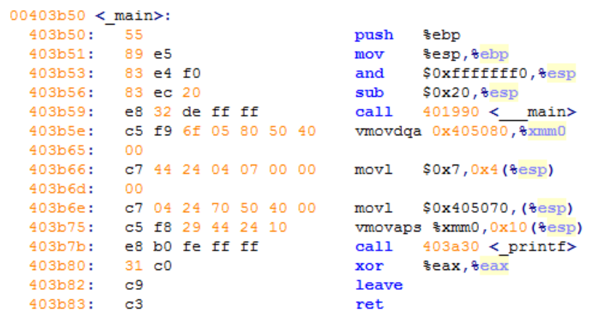
```
gcc -march=native -mavx main.c -o avx.o -O3
objdump -d avx.o > avx.txt

gcc main.c -o noavx.o
objdump -d noavx.o > noavx.txt
```



```
#include <stdio.h>
#define V_SIZE 4

int main(void)
{
    int a[V_SIZE] = {1, 2, 3, 4};
    int b[V_SIZE] = {7, 8, 9, 10};
    int c[V_SIZE];
    int i;

    for (i = 0; i < V_SIZE; i++) {
        c[i] = a[i] * b[i];
    }

    printf(" %d ", c[0]);

    return 0;
}
```

Fig. 3. Example program

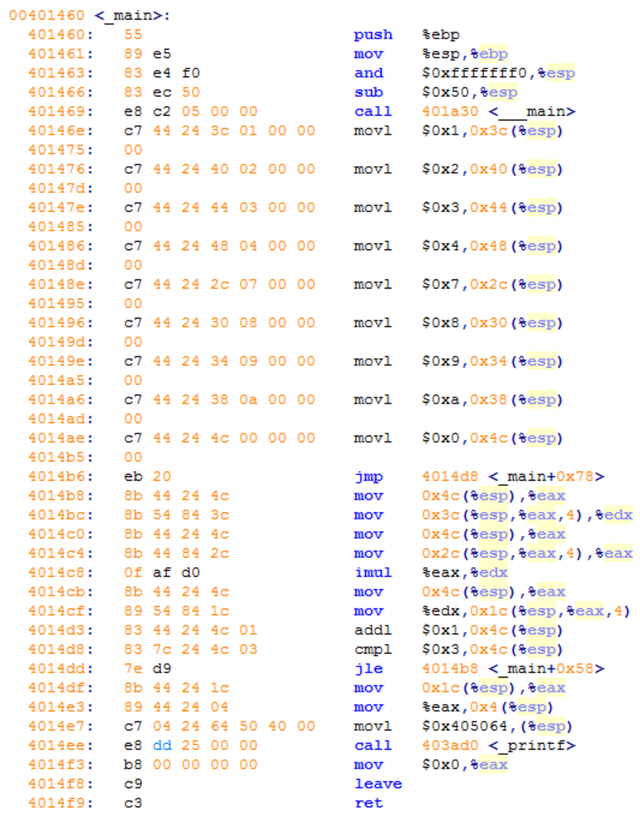
The assembled code with without AVX instructions are Fig. 4. and Fig. 5., respectively:



```
00403b50 <_main>:
  403b50:  55                      push   %ebp
  403b51:  89 e5                   mov    %esp,%ebp
  403b53:  83 e4 f0                and    $0xfffffff0,%esp
  403b56:  83 ec 20                sub    $0x20,%esp
  403b59:  e8 32 de ff ff          call   401990 <___main>
  403b5e:  c5 f9 6f 05 80 50 40    vmovdqa 0x405080,%xmm0
  403b65:  00
  403b66:  c7 44 24 04 07 00 00    movl   $0x7,0x4(%esp)
  403b6d:  00
  403b6e:  c7 04 24 70 50 40 00    movl   $0x405070,(%esp)
  403b75:  c5 f8 29 44 24 10       vmovaps %xmm0,0x10(%esp)
  403b7b:  e8 b0 fe ff ff          call   403a30 <_printf>
  403b80:  31 c0                   xor    %eax,%eax
  403b82:  c9                      leave
  403b83:  c3                      ret
```

Fig. 4. Main compiled with AVX instuctions.

As it can be seen in Fig. 4. and Fig. 5., the main function in the version compiled with AVX instructions (Fig.4.) has way fewer lines of code, due to the AVX optimisations.



```
00401460 <_main>:
  401460:  55                      push   %ebp
  401461:  89 e5                   mov    %esp,%ebp
  401463:  83 e4 f0                and    $0xfffffff0,%esp
  401466:  83 ec 50                sub    $0x50,%esp
  401469:  e8 c2 05 00 00          call   401a30 <___main>
  40146e:  c7 44 24 3c 01 00 00    movl   $0x1,0x3c(%esp)
  401475:  00
  401476:  c7 44 24 40 02 00 00    movl   $0x2,0x40(%esp)
  40147d:  00
  40147e:  c7 44 24 44 03 00 00    movl   $0x3,0x44(%esp)
  401485:  00
  401486:  c7 44 24 48 04 00 00    movl   $0x4,0x48(%esp)
  40148d:  00
  40148e:  c7 44 24 2c 07 00 00    movl   $0x7,0x2c(%esp)
  401495:  00
  401496:  c7 44 24 30 08 00 00    movl   $0x8,0x30(%esp)
  40149d:  00
  40149e:  c7 44 24 34 09 00 00    movl   $0x9,0x34(%esp)
  4014a5:  00
  4014a6:  c7 44 24 38 0a 00 00    movl   $0xa,0x38(%esp)
  4014ad:  00
  4014ae:  c7 44 24 4c 00 00 00    movl   $0x0,0x4c(%esp)
  4014b5:  00
  4014b6:  eb 20                   jmp    4014d8 <_main+0x78>
  4014b8:  8b 44 24 4c             mov    0x4c(%esp),%eax
  4014bc:  8b 54 84 3c             mov    0x3c(%esp,%eax,4),%edx
  4014c0:  8b 44 24 4c             mov    0x4c(%esp),%eax
  4014c4:  8b 44 84 2c             mov    0x2c(%esp,%eax,4),%eax
  4014c8:  0f af d0                imul   %eax,%edx
  4014cb:  8b 44 24 4c             mov    0x4c(%esp),%eax
  4014cf:  89 54 84 1c             mov    %edx,0x1c(%esp,%eax,4)
  4014d3:  83 44 24 4c 01          addl   $0x1,0x4c(%esp)
  4014d8:  83 7c 24 4c 03          cmpl   $0x3,0x4c(%esp)
  4014dd:  7e d9                   jle    4014b8 <_main+0x58>
  4014df:  8b 44 24 1c             mov    0x1c(%esp),%eax
  4014e3:  89 44 24 04             mov    %eax,0x4(%esp)
  4014e7:  c7 04 24 64 50 40 00    movl   $0x405064,(%esp)
  4014ee:  e8 dd 25 00 00          call   403ad0 <_printf>
  4014f3:  b8 00 00 00 00          mov    $0x0,%eax
  4014f8:  c9                      leave
  4014f9:  c3                      ret
```

Fig. 5. Main compiled without AVX instuctions.

The version without AVX instructions needs 32 lines of assembly code to complete the task of multiplying the vectors (from 40149e to 4014e7), while the version with AVX instructions only needs 6 lines (from 403b5e to 403b75) to execute the same task.

This is due to the use of the AVX instructions `vmovdqa`, used to move aligned packed integer values; and `vmovaps`, used to move aligned packed single-precision floating-point values.

References

[1] M. Flynn. (2011). *Flynn's Taxonomy*. In: D. Padua (eds) Encyclopedia of Parallel Computing. Springer, Boston, MA. Retrieved from *https://doi.org/10.1007/978-0-387-09766-4_2*

[2] M. J. Flynn. (1966). *Very High-Speed Computing Systems*. In: Proceedings of the IEEE, vol. 54, No. 12, December 1966. pp. 1901–1909.

[3] R. Oshana. (2016). *Multicore Software Development Techniques - Applications, Tips and Tricks*. Newnes.

[4] D. A. Patterson and J. L. Hennessy. (2005). *Computer Organization and Design - The Hardware/Software Interface Third Edition*. Elsevier.

[5] D. A. Patterson and J. L. Hennessy. (2012). *Computer Architecture - A Quantitative Approach Fifth Edition*. Elsevier.

[6] W. M. Hwu. (2011). *Superscalar Processors*. In: D. Padua (eds) Encyclopedia of Parallel Computing. Springer, Boston, MA. Retrieved from *https://doi.org/10.1007/978-0-387-09766-4_280*

[7] A. A. Alipour, A. Aminpour, and E. Ansari. (2018) *Parallel PatchMatch using Open-MP Extension*. Conference: 5th International Conference Electrical engineering and computer With emphasis on native knowledge.

[8] Motorola. (1994). *M68060 User's Manual - Including the MC68060, MC68LC060, and MC68EC060*. Retrieved from *https://www.nxp.com/docs/en/data-sheet/MC68060UM.pdf*

[9] Intel. (1997). *Pentium® Processor with MMX™ Technology*. Retrieved from *http://datasheets.chipdb.org/Intel/x86/Pentium%20MMX/24318504.PDF*

[10] International Business Machines. (2008). *IBM PowerPC 970FX RISC Microprocessor User's Manual - Version 1.7*. Retrieved from *https://web.archive.org/web/20140522012345/https://www-01.ibm.com/chips/techlib/techlib.nsf/techdocs/DC3D43B729FDAD 2C00257419006FB955/$file/970FX_user_manual.v1.7.2008MAR14_pub.pdf*

[11] Intel. (2021). *8th and 9th Generation Intel® Core™ Processor Families and Intel® Xeon®E Processor Families - Datasheet, Volume 1 of 2*. Retrieved from: *https://www.intel.com/content/www/us/en/products/docs/processors/core/core-technical-resources.html*

[12] Y. Le. (2008). *Schema Validation with Intel® Streaming SIMD Extensions 4 (Intel® SSE4)*. Retrieved from: *https://software.intel.com/content/www/us/en/develop/articles/schema-validation-with-intel-streaming-simd-extensions-4-intel-sse4.html*

[13] Oracle. (2014). *x86 Assembly Language Reference Manual*. Retrieved from: *https://docs.oracle.com/cd/E36784_01/pdf/E36859.pdf*