



Instytut Informatyki Politechniki Śląskiej  
Zespół Mikroinformatyki i Teorii Automatów  
Cyfrowych

## Laboratorium JA



| Rok akademicki  | Rodzaj studiów*: SSI/NSI/NSM | Numer ćwiczenia:     | Grupa     | Sekcja   |
|---|------------------------------|----------------------|-----------|----------|
| <b>2017/2018</b>  | <b>SSI</b>                   | <b>5</b>             | <b>6</b>  | <b>1</b> |
| Data i godzina planowana ćwiczenia:<br>dd/mm/rrrr - gg:mm | 27/03/2018-11:45             | Prowadzący:<br>OA/AO | <b>OA</b> |          |
| Data i godzina wykonania ćwiczenia:<br>dd/mm/rrrr - gg:mm | 27/03/2018-11:45             |                      |           |          |

## *Sprawozdanie*

Temat ćwiczenia:

**Zapoznanie się z instrukcjami  
MMX i SSE w j. asemblera.**

**Skład sekcji:**

1. Bartłomiej Krasoń

# Cel

Celem ćwiczenia jest poznanie działania rozkazów MMX i SSE wykorzystywanych do wykonywania precyzyjnych obliczeń zmiennoprzecinkowych procesorów x86 firmy Intel dla środowiska Windows.  
~Źródło – Instrukcja ćwiczenia LAB5

# Rozwiązanie

Korzystając z kodu zamieszczonego w instrukcji sprawdziłem czy mój procesor obsługuje instrukcje MMX, SSE i SSE2:

```
C:\Users\Bartek\Desktop\JA\Lab5\Project1\Debug\Project1.exe
The following features are supported : n      SSE3
MONITOR / MWAIT
tCPL Qualified Debug Store
tVirtual Machine Extensions
tEnhanced Intel SpeedStep Technology
Thermal Monitor 2
tSupplemental Streaming SIMD Extensions 3
L1 Context ID
CMPXCHG16B Instruction
xTPR Update Control
Perf\Debug Capability MSR
SSE4.1 Extensions
SSE4.2 Extensions
POPCNT Instruction
x87 FPU On Chip
Virtual-8086 Mode Enhancement
Debugging Extensions
Page Size Extensions
Time Stamp Counter
RDMSR and WRMSR Support
Physical Address Extensions
Machine Check Exception
CMPXCHG8B Instruction
APIC On Chip
SYSENTER and SYSEXIT
Memory Type Range Registers
PTE Global Bit
Machine Check Architecture
Conditional Move/Compare Instruction
Page Attribute Table
36-bit Page Size Extension
CFLUSH Extension
Debug Store
Thermal Monitor and Clock Ctrl
MMX Technology
FXSAVE/FXRSTOR
SSE Extensions
SSE2 Extensions
Self Snoop
Multithreading Technology
Thermal Monitor
Pending Break Enable
LAHF / SAHF in 64-bit mode
LZCNT instruction
PREFETCH and PREFETCHW Instructions
RDTSCP instruction
64 bit Technology
nCPU Brand String : Intel(R) Core(TM) i5-6200U CPU @ 2.30GHz
Cache Line Size = 64
L2 Associativity = 6
Cache Size = 256K

Number of Cores = 8
```

Następnie w asemblerze zaimplementowałem następujące procedury:

- 1) RadToDegAsm – która konwertuje radiany na stopnie

```
1  .686
2  .xmm
3  .model flat, stdcall
4  .data
5  tmp_deg real8 0.0
6  n_180 real8 180.0
7  M_PI real8 3.141592653589793238
8  .code
9  RadToDegAsm PROC rad: QWORD ;parametr w radianach
10  movq xmm0, n_180 ; wczytanie stałej
11  movq xmm1, M_PI ; wczytanie PI
12  divpd xmm0, xmm1 ; dzielenie
13  movq xmm3, xmm0 ; zapamiętanie
14
15  movq xmm0, rad ; wczytanie radianów
16  mulpd xmm0, xmm3 ; mnożenie rad*180/pi
17
18  movlpd tmp_deg, xmm0 ; zapamiętanie
19
20  fld tmp_deg ; zwrócenie wyniku w stopniach
21  ret
22 RadToDegAsm ENDP
```

- 2) check\_for\_mmx\_asm – która sprawdza czy procesor obsługuje rozszerzenie MMX

```
24  check_for_MMX_asm PROC
25  xor eax, eax
26  cpuid
27  test eax, eax
28  jz Failure
29  mov eax, 1
30  cpuid
31  shr edx, 23
32  and edx, 1
33  mov eax, edx
34  ret
35  Failure:
36  ret
37  check_for_MMX_asm ENDP
```

- 3) check\_for\_sse\_asm – która sprawdza czy procesor obsługuje rozszerzenie SSE

```
39  check_for_SSE_asm PROC
40  xor eax, eax
41  cpuid
42  test eax, eax
43  jz Failure
44  mov eax, 1
45  cpuid
46  shr edx, 25
47  and edx, 1
48  mov eax, edx
49  ret
50  Failure: ret
51  check_for_SSE_asm ENDP
```

Następnie przetestowałem działanie powyższych procedur w projekcie w języku C++ tak jak to robiliśmy na wcześniejszych zajęciach:

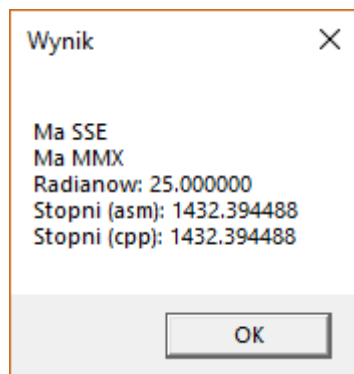
```
// TODO: Place code here.
std::string text = "";
bool check_sse = check_for_SSE_asm();
bool check_mmx = check_for_MMX_asm();
if (check_sse) text += "Ma SSE\n";
else text += "Nie ma SSE\n";
if (check_mmx) text += "Ma MMX\n";
else text += "Nie ma MMX\n";
double rad = 25.0;
text += "Radianow: ";
text += std::to_string(rad);
double degrees_result_asm = RadToDegAsm(rad);
text += "\nStopni (asm): ";
text += std::to_string(degrees_result_asm);
double degrees_result_cpp = RadToDegNMEACpp(rad, bSSE2)/100;
text += "\nStopni (cpp): ";
text += std::to_string(degrees_result_cpp);

std::wstring stemp = s2ws(text);
LPCWSTR result = stemp.c_str();

MessageBox(NULL, result, L"Wynik", MB_OK);
```

W celu porównania wyniku funkcji obliczającej stopnie w assemblerze, skorzystałem z funkcji RadToDegNMEACpp zamieszczonej w instrukcji , która oblicza stopnie na podstawie radianów jednakże w formacie NMEA, czyli aby uzyskać odpowiedni wynik, należy podzielić go przez 100.

Wynik pracy programu:



## Wnioski

Wykonując to ćwiczenie poznałem nowe zestawy instrukcji dla procesora, takie jak MMX, SSE, SSE2. Znajomość takich zestawów okazuje się bardzo przydatna, gdyż dzięki nim, pisząc nasze programy możemy zwiększyć dokładność wykonywanych obliczeń, co wpływa pozytywnie na poprawność i funkcjonalność naszych aplikacji. Jednakże trzeba też uważać na kwestię tego, że nie wszystkie procesory mogą obsługiwać te właśnie zestawy instrukcji. Dlatego dowiedziałem się również, jak z poziomu programu assemblerowego czy to napisanego w języku C++ można sprawdzić, jakimi zestawami instrukcji włada nasz procesor.