Instytut Informatyki Politechniki Śląskiej
Zespół Mikroinformatyki i Teorii Automatów Cyfrowych

Laboratorium JA

Rok akademicki	Rodzaj studiów*: SSI/NSI/NSM	Numer ćwiczenia:	Grupa	Sekcja
2017/2018	**SSI**	**5**	**6**	**1**

Data i godzina planowana ćwiczenia: dd/mm/rrrr - gg:mm	**27/03/2018-11:45**	**Prowadzący**: OA/AO	**OA**
Data i godzina wykonania ćwiczenia: dd/mm/rrrr - gg:mm	**27/03/2018-11:45**		

Sprawozdanie

Temat ćwiczenia:

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

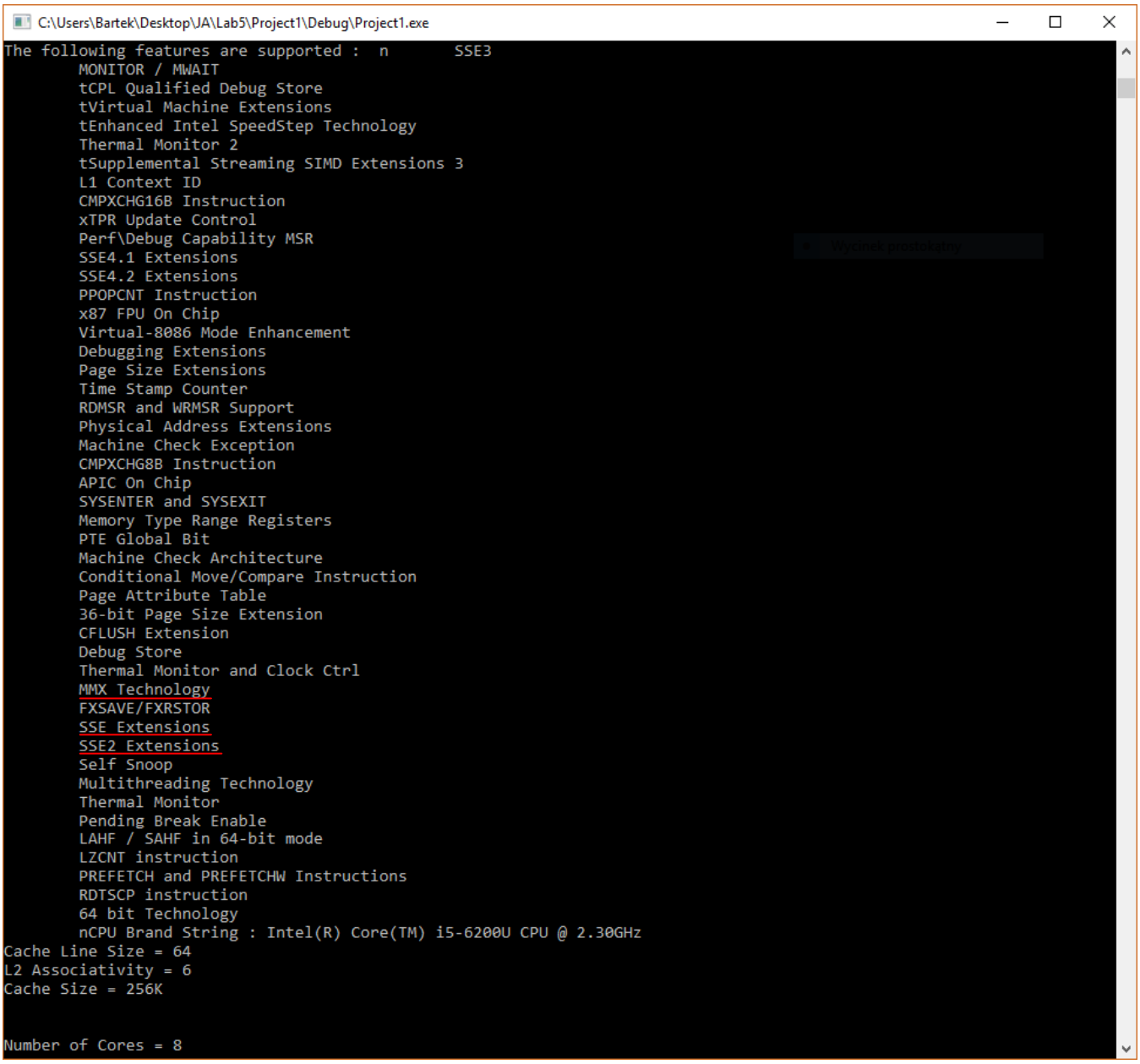
Skład sekcji:	1.Bartłomiej Krasoń

Cel

Celem ćwiczenia jest poznanie działania rozkazów MMX i SEE 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:



Następnie w asemblerze zaimplementowałem następujące procedury:

1) RedToDegAsm – która konwertuje radiany na stopnie

```
.686
.xmm
.model flat, stdcall
.data
tmp_deg real8 0.0
n_180 real8 180.0
M_PI real8 3.141592653589793238
.code
RadToDegAsm PROC rad: QWORD ;parametr w radianach
 movq xmm0, n_180 ; wczytanie stałej
 movq xmm1, M_PI ; wczytanie PI
 divpd xmm0, xmm1 ; dzielenie
 movq xmm3, xmm0 ; zapamiętanie

 movq xmm0, rad ; wczytanie radianów
 mulpd xmm0, xmm3 ; mnożenie rad*180/pi

 movlpd tmp_deg, xmm0 ; zapamiętanie

 fld tmp_deg ; zwrócenie wyniku w stopniach
 ret
RadToDegAsm ENDP
```

2) check_for_mmx_asm – która sprawdza czy procesor obsługuje rozszerzenie MMX

```
check_for_MMX_asm PROC
 xor eax, eax
 cpuid
 test eax, eax
 jz Failure
 mov eax, 1
 cpuid
 shr edx, 23
 and edx, 1
 mov eax, edx
 ret
 Failure:
 ret
 check_for_MMX_asm ENDP
```

3) check_for_sse_asm – która sprawdza czy procesor obsługuje rozszerzenie SSE

```
check_for_SSE_asm PROC
xor eax, eax
cpuid
test eax, eax
jz Failure
mov eax, 1
cpuid
shr edx, 25
and edx, 1
mov eax, edx
ret
Failure: ret
check_for_SSE_asm ENDP
```

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

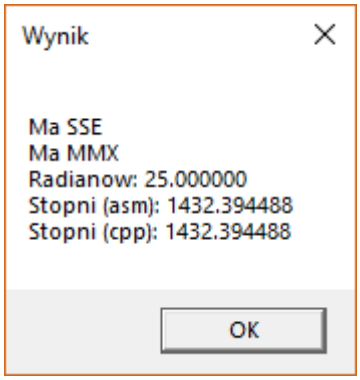
```
// TODO: Place code here.
std::string text = "";
bool check_sse = check_for_SSE_asm();
bool check_mmx = check_for_MMX_asm();
if (check_sse) text += "Ma SSE\n";
else text += "Nie ma SSE\n";
if (check_mmx) text += "Ma MMX\n";
else text += "Nie ma MMX\n";
double rad = 25.0;
text += "Radianow: ";
text += 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 asemblerze, 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 asemblerowego czy to napisanego w języku C++ można sprawdzić, jakimi zestawami instrukcji włada nasz procesor.