Projekt: Kodowanie ROT13

Dominik Dziembała

Założenia projektu

- Szyfr ROT13 jest szyfrem przesuwającym.
- Jego działanie polega na zamianie każdego znaku alfabetu łacińskiego na znak występujący 13 pozycji po nim.
- Wielkość liter nie ma znaczenia, czyli wielka litera jest zamieniana na wielką literę, mała na małą.
- Szyfr ROT13 jest swoją własną funkcją odwrotną.

Przykład tekstu przed i po

- Przed:
  ABCDEFGHIJKLMNOPQRSTUVWXYZ 1234567890
  abcdefghijklmnopqrstuvwxyz
- Po:
  NOPQRSTUVWXYZABCDEFGHIJKLM 1234567890
  nopqrstuvwxyzabcdefghijklm

Analiza projektu

- W momencie przekroczenia zakresu alfabetu odliczanie kolejnych liter kontynuujemy od litery A/a.
- Można zauważyć, że w momencie przekroczenia zakresu podczas przesuwania się po kolejnych literach w prawo o 13 możemy wybrać literę o 13 pozycji wcześniej.
- Wyznaczenie litery granicznej, do wystąpienia której będzie się dodawać 13, a od jej wystąpienia odejmować.
- Literą graniczną jest N/n.

Utworzenie wątków

```
private void prepareThreads(string[] tab, int id)
        {
            if (arrayOfThreads.Length != cores)
            {
                Array.Resize<Thread>(ref arrayOfThreads, cores);
            }
            progressbars[id].Value = 10;
            for (int i = 0; i < cores; i++)
            {
                arrayOfThreads[i] = new Thread(new ParameterizedThreadStart(runThreads));
            }
            watches[id].Start();
            for (int i = 0; i < cores; i++)
            {
                object arguments = new object[3] { i, tab, id};
                arrayOfThreads[i].Start(arguments);
            }
            for (int i = 0; i < cores; i++)
            {
                arrayOfThreads[i].Join();
            }
            watches[id].Stop();
        }
```

Implementacja w C

```
void __stdcall cEncodeText(char* text, char* text2, int startIndex, int endIndex)
{
    int j = 0;
    for (int i = startIndex; i < endIndex; i++)
    {
        if ((text[i] >= 65) && (text[i] <= 90))
        {
            if (text[i] > 77)
            {
                text2[j] = text[i] - 13;
            }
            else
            {
                text2[j] = text[i] + 13;
            }
        }
```

Implementacja w C c.d.

```
else if ((text[i] >= 97) && (text[i] <= 122))
{
    if (text[i] > 109)
    {
        text2[j] = text[i] - 13;
    }
    else
    {
        text2[j] = text[i] + 13;
    }
}
else
{
    text2[j] = text[i];
}
j++;
}
}
```

Implementacja wektorowa w ASM

- .data
- characterz db 91, 0, 91, 0, 91, 0, 91, 0, 91, 0, 91, 0, 91, 0, 91, 0
- charactersmallz db 123, 0, 123, 0, 123, 0, 123, 0, 123, 0, 123, 0, 123, 0, 123, 0
- charactera db 65, 0, 65, 0, 65, 0, 65, 0, 65, 0, 65, 0, 65, 0, 65, 0
- charactersmalla db 97, 0, 97, 0, 97, 0, 97, 0, 97, 0, 97, 0, 97, 0, 97, 0
- characterm db 78, 0, 78, 0, 78, 0, 78, 0, 78, 0, 78, 0, 78, 0, 78, 0
- charactersmallm db 110, 0, 110, 0, 110, 0, 110, 0, 110, 0, 110, 0, 110, 0, 110, 0
- subnumber db 255, 0, 255, 0, 255, 0, 255, 0, 255, 0, 255, 0, 255, 0, 255, 0
- addnumber dw 510, 510, 510, 510, 510, 510, 510, 510
- number db 13, 0, 13, 0, 13, 0, 13, 0, 13, 0, 13, 0, 13, 0, 13, 0
- negativenumber dw -13, -13, -13, -13, -13, -13, -13, -13

Implementacja wektorowa w ASM c.d.

- .code
- vectorialEncodeText proc
  - ;push using registers to the stack
  - push rcx
  - push rdi
  - push r13
  - ;initialise rdi as text jumper and copy amount of characters to r13
  - mov r13, rdx
  - mov rdi, 0
  - ;check if text is not empty
  - cmp r13, 0
  - je _end

Implementacja wektorowa w ASM c.d.

```
;clear xmm registers
_loop:  xorps xmm1, xmm1
xorps xmm2, xmm2
xorps xmm3, xmm3
xorps xmm4, xmm4
xorps xmm5, xmm5
xorps xmm6, xmm6
;load 8 characters to xmm7
movups xmm7, [rcx+rdi]
;load test characters to registers xmm1-xmm6
paddb xmm1, characterz
paddb xmm2, charactersmallz
paddb xmm3, charactera
paddb xmm4, charactersmalla
paddb xmm5, characterm
paddb xmm6, charactersmallm
;check if character in xmm7 is greater than character in xmm1-xmm6
pcmpgtb xmm1, xmm7
pcmpgtb xmm2, xmm7
pcmpgtb xmm3, xmm7
pcmpgtb xmm4, xmm7
pcmpgtb xmm5, xmm7
pcmpgtb xmm6, xmm7
```

Implementacja wektorowa w ASM c.d.

```
;check small letter, if it is - value equals 510 or 255
paddsw xmm2, xmm4
paddsw xmm2, xmm6
;clear xmm4 and xmm6
xorps xmm4, xmm4
xorps xmm6, xmm6
;load to xmm4 510, to xmm6 255 and compare it with small letters test
paddsw xmm4, addnumber
paddsb xmm6, subnumber
pcmpeqb xmm4, xmm2
pcmpeqb xmm6, xmm2
;load +13 to xmm4 values if results of comparsion equal -1
xorps xmm2, xmm2
paddb xmm2, number
pand xmm4, xmm2
;load -13 to xmm6 values if results of comparsion equal -1
xorps xmm2, xmm2
paddw xmm2, negativenumber
pand xmm6, xmm2
;store operation on small letters in xmm4
paddsw xmm4, xmm6
```

Implementacja wektorowa w ASM c.d.

```
;check capital letter, if it is - value is 510 or 255
paddsw xmm1, xmm3
paddsw xmm1, xmm5
;clear xmm3 and xmm5
xorps xmm3, xmm3
xorps xmm5, xmm5
;load to xmm3 510, to xmm5 255 and compare it with capital letters test
paddsw xmm3, addnumber
paddsb xmm5, subnumber
pcmpeqb xmm3, xmm1
pcmpeqw xmm5, xmm1
;load +13 to xmm3 values if results of comparsion equal -1
xorps xmm1, xmm1
paddb xmm1, number
pand xmm3, xmm1
;load -13 to xmm5 values if results of comparsion equal -1
xorps xmm1, xmm1
paddw xmm1, negativenumber
pand xmm5, xmm1
;store operation on capital letters in xmm3
paddsw xmm3, xmm5
```

Implementacja wektorowa w ASM c.d.

- ;sum operations on capital and small letters in xmm3
- paddsw xmm3, xmm4
- ;add xmm3 to xmm7, copy it to text array and move to the next 8 characters
- paddsw xmm7, xmm3
- movups [rcx+rdi], xmm7
- add rdi, 16
- ;check if text has still any characters
- cmp r13, 8
- jbe _end
- ;decrease amount of characters and jump to begin of loop
- sub r13, 8
- jmp _loop

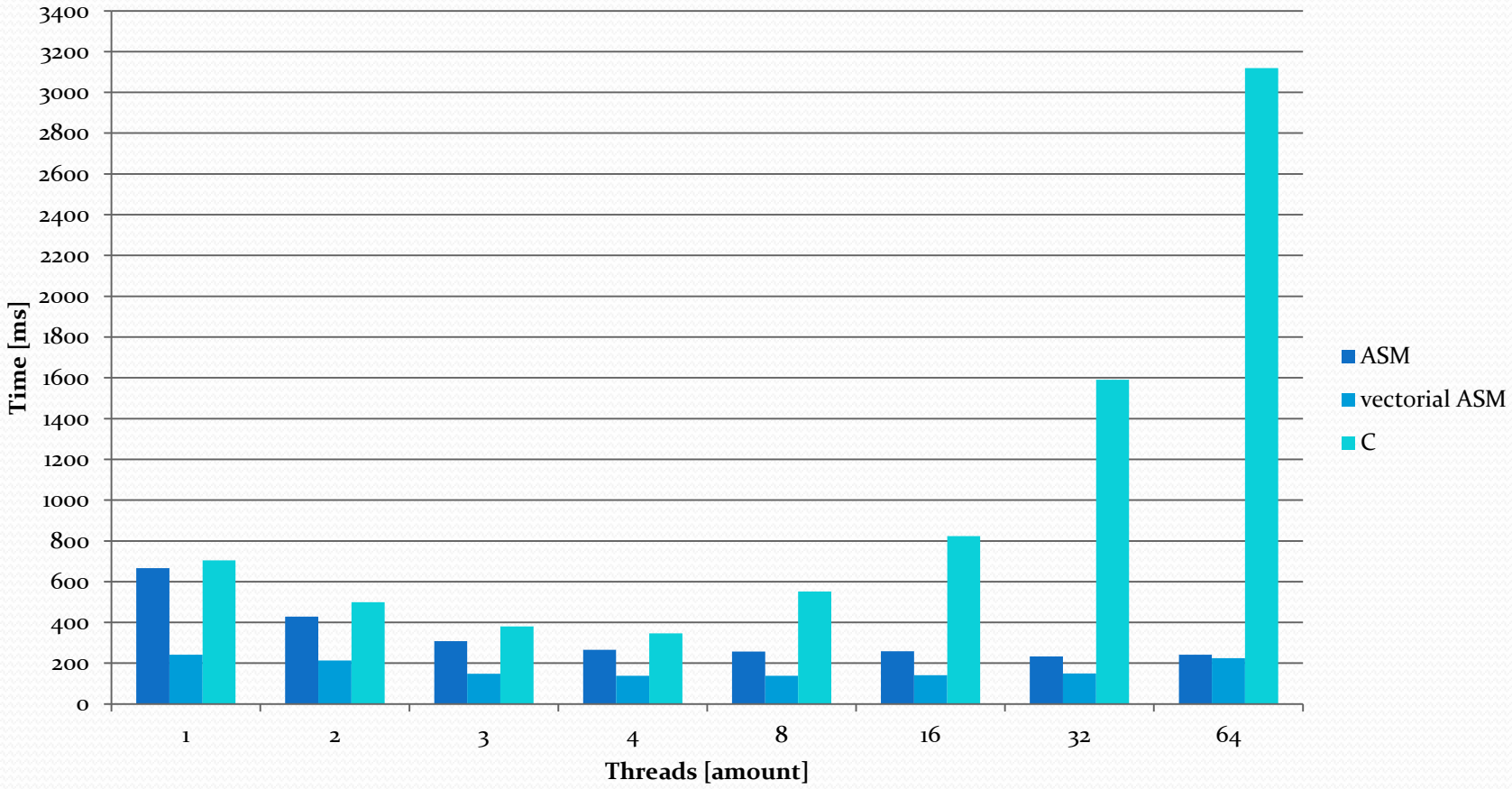
Implementacja wektorowa w ASM c.d.

- ;get values of using register from the stack and return encoded text
- _end: pop r13
- pop rdi
- mov rax, rcx
- pop rcx
- ret
- vectorialEncodeText endp

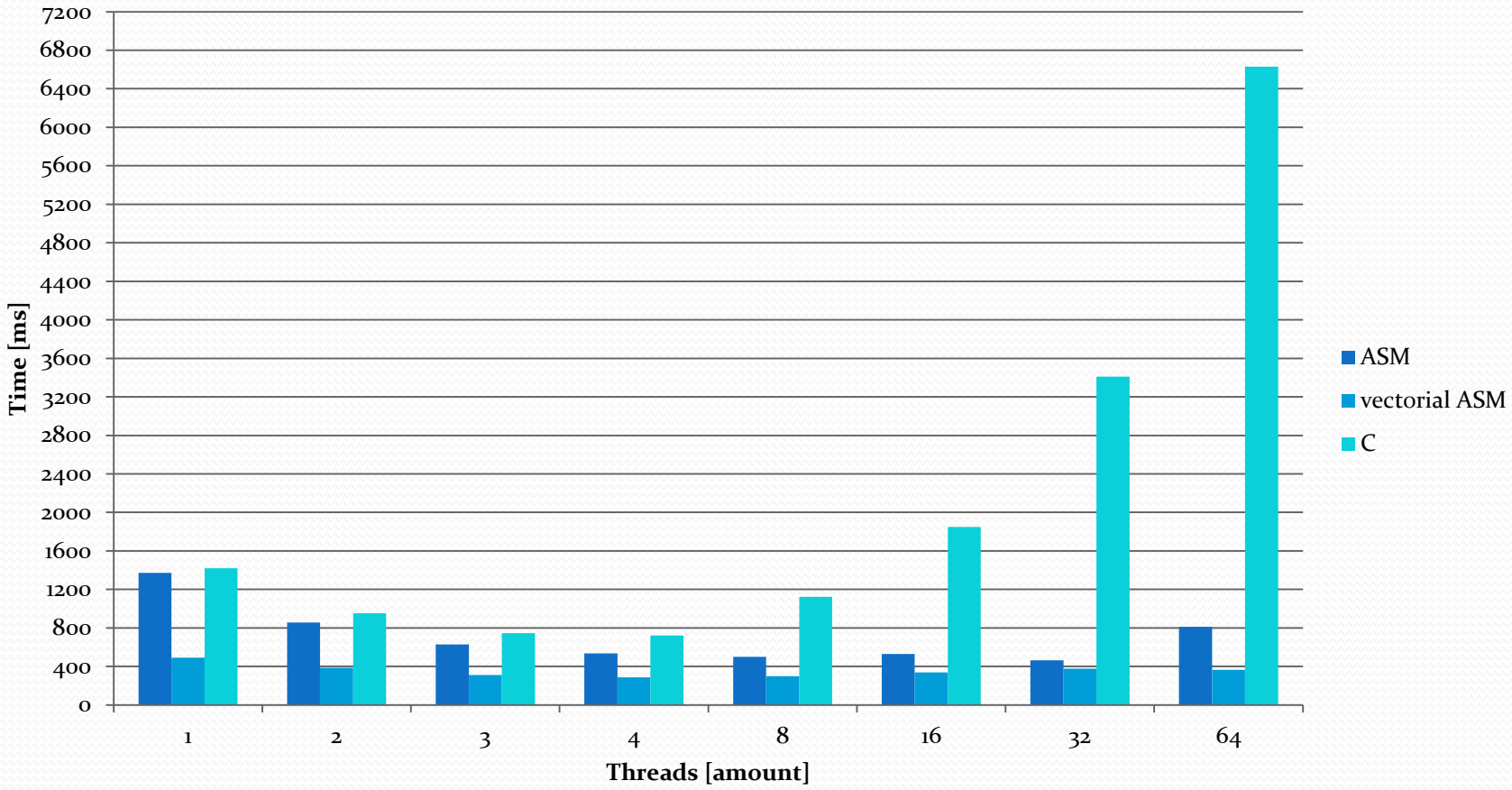
Procesor

- Intel Core i5-2540M
- Liczba rdzeni: 2
- Liczba wątków: 4
- Bazowa częstotliwość: 2,60 GHz
- Maksymalna częstotliwość 3,30 GHz
- SSE4

Wykresy czasowe dla pliku ok. 64MB



Wykresy czasowe dla pliku ok. 128MB



Dziękuję za uwagę!