

AKOmpañiment

[PATCH 1] 2017-11-20

SLAJDY: [HTTP://AKO.DRAGONIC.EU](http://ako.dragonic.eu)

REJESTRY

EAX		EBX		ECX		EDX
ESI		EDI		ESP		EBP
EFLAGS		EIP		...?		

REJESTRY

EAX		EBX		ECX		EDX
ESI		EDI		ESP		EBP
EFLAGS		EIP		...		?

ARSENAL

NOP, MOV, XCHG, LEA, PUSH{F/A}, POP{F/A}
STC, CLC, CMC, CBW, CWDE, CWD, CMP, BT{S/R/C}
ADD, ADC, SUB, SBB, {I}DIV, {I}MUL, INC, DEC
SAL/R, SHL/R, ROL/R, RCL/R
NEG, NOT, AND, OR, XOR
JMP, J???, CALL, RET

ARSENAL

FINIT, FLD, FILD, FLD{Z/1/PI/L2E/L2T/LG2/LN2}
F{I}ADD{P}, F{I}SUB{P}, F{I}MUL{P}, F{I}DIV{P}
FSQRT, FSIN, FCOS, FABS, F2XM1, FRNDINT
FST{P}, FIST{P}, FXCH, FCHS, FCOMI

NOP • NO OPERATION

~_(ツ)_/~

ZAJMUJE 1 BAJT, 0x90

MOV • MOVE DATA

MOV A, B

A = WARTOŚĆ B

MOV • MOVE DATA

MOV EAX, EBX

EAX = WARTOŚĆ EBX

MOV • MOVE DATA

```
MOV EAX, [EDX + 2*ESI + 0F420h]
```

EAX = WARTOŚĆ POD TYM ADRESEM ^

SIB • SCALE, INDEX, BASE

ZASADY DLA: $[EDX + 2 * ESI + 0F420h]$

JEDEN REJESTR BEZ MNOŻENIA

INDEX: DRUGI REJESTR * 1/2/4/8 (SCALE)

BASE, TJ WYBRANA STAŁA

NIE WOLNO UŻYWAĆ ESP

TYLKO DODAWANIE

LEA • LOAD EFFECTIVE ADDRESS

LEA EAX, [EDX + 2*ESI + 0F420h]

EAX = EDX + 2*ESI + 0F420h

MOV • MOVE DATA

MOV [adres], 80h

NA CO WSKAZUJE ADRES?

MOV • MOVE DATA

MOV BYTE PTR [adres], 80h

WORD PTR 0080h

DWORD PTR 00000080h

QWORD PTR ...00000080h

MOV • MOVE DATA

ADRES: 0000F438h

MOV BYTE PTR [ADRES], 80h

0000F410	02	00	03	00	01	00	00	00	30	84	04	08	34	00	00	00
0000F420	ec	22	00	00	00	00	00	00	34	00	20	00	08	00	28	00
0000F430	ec	22	00	00	00	00	00	00	34	00	20	00	08	00	28	00
0000F440	ec	22	00	00	00	00	00	00	34	00	20	00	08	00	28	00

MOV • MOVE DATA

ADRES: 0000F438h

MOV BYTE PTR [ADRES], 80h

0000F410	02	00	03	00	01	00	00	00	30	84	04	08	34	00	00	00
0000F420	ec	22	00	00	00	00	00	00	34	00	20	00	08	00	28	00
0000F430	ec	22	00	00	00	00	00	00	80	00	20	00	08	00	28	00
0000F440	ec	22	00	00	00	00	00	00	34	00	20	00	08	00	28	00

MOV • MOVE DATA

ADRES: 0000F438h

MOV WORD PTR [ADRES], 80h

0000F410	02	00	03	00	01	00	00	00	30	84	04	08	34	00	00	00
0000F420	ec	22	00	00	00	00	00	00	34	00	20	00	08	00	28	00
0000F430	ec	22	00	00	00	00	00	00	34	00	20	00	08	00	28	00
0000F440	ec	22	00	00	00	00	00	00	34	00	20	00	08	00	28	00

MOV • MOVE DATA

ADRES: 0000F438h

MOV DWORD PTR [ADRES], 80h

0000F410	02	00	03	00	01	00	00	00	30	84	04	08	34	00	00	00
0000F420	ec	22	00	00	00	00	00	00	34	00	20	00	08	00	28	00
0000F430	ec	22	00	00	00	00	00	00	34	00	20	00	08	00	28	00
0000F440	ec	22	00	00	00	00	00	00	34	00	20	00	08	00	28	00

MOV • MOVE DATA

ADRES: 0000F438h

MOV DWORD PTR [ADRES], 80h

0000F410	02	00	03	00	01	00	00	00	30	84	04	08	34	00	00	00
0000F420	ec	22	00	00	00	00	00	00	34	00	20	00	08	00	28	00
0000F430	ec	22	00	00	00	00	00	00	80	00	00	00	08	00	28	00
0000F440	ec	22	00	00	00	00	00	00	34	00	20	00	08	00	28	00

XCHG • EXCHANGE

XCHG A, B

ZAMIEŃ WARTOŚĆ $A \leftrightarrow B$

NIE WOLNO UŻYWAĆ STAŁYCH

FLAGI • SET/CLEAR/COMPLEMENT

STC \rightarrow CF = 1

CLC \rightarrow CF = 0

CMC \rightarrow CF = !CF

POP/PUSH • OPERACJE NA STOSIE

PUSH EAX

POP DX

POP AX

PUSH DWORD PTR 42

POP EAX

ESP: 00FF563A

EAX: 5204F36B

EDX: 00000000

00

00

00

00

00

00

64

5F

ESP

POP/PUSH • OPERACJE NA STOSIE



PUSH EAX

POP DX

POP AX

PUSH DWORD PTR 42

POP EAX

ESP: 00FF563A

EAX: 5204F36B

EDX: 00000000

00

00

00

00

00

00

64

5F



ESP

POP/PUSH • OPERACJE NA STOSIE



PUSH EAX

POP DX

POP AX

PUSH DWORD PTR 42

POP EAX

ESP: 00FF5636

EAX: 5204F36B

EDX: 00000000

00

00

6B

F3

04

52

64

5F



POP/PUSH • OPERACJE NA STOSIE

PUSH EAX

ESP: 00FF5636

00

00

EIP → POP DX

6B

← ESP

POP AX

F3

PUSH DWORD PTR 42

EAX: 5204F36B

04

52

POP EAX

EDX: 00000000

64

5F

POP/PUSH • OPERACJE NA STOSIE

PUSH EAX

ESP: 00FF5638

00

EIP → POP DX

00

6B

POP AX

F3

PUSH DWORD PTR 42

EAX: 5204F36B

04 ← ESP

52

POP EAX

EDX: 0000F36B

64

5F

POP/PUSH • OPERACJE NA STOSIE

PUSH EAX

POP DX

 POP AX

PUSH DWORD PTR 42

POP EAX

ESP: 00FF5638

EAX: 5204F36B

EDX: 0000F36B

00

00

6B

F3

04

52

64

5F

 ESP

POP/PUSH • OPERACJE NA STOSIE

PUSH EAX

POP DX

 POP AX

PUSH DWORD PTR 42

POP EAX

ESP: 00FF563A

EAX: 52045204

EDX: 0000F36B

00

00

6B

F3

04

52

64

5F

 ESP

POP/PUSH • OPERACJE NA STOSIE

PUSH EAX

POP DX

POP AX

 PUSH DWORD PTR 42

POP EAX

ESP: 00FF563A

EAX: 52045204

EDX: 0000F36B

00

00

6B

F3

04

52

64

5F

← ESP

POP/PUSH • OPERACJE NA STOSIE

PUSH EAX

POP DX

POP AX

 PUSH DWORD PTR 42

POP EAX

ESP: 00FF5636

EAX: 52045204

EDX: 0000F36B

00

00

42

00

00

00

64

5F

← ESP

POP/PUSH • OPERACJE NA STOSIE

PUSH EAX

POP DX

POP AX

PUSH DWORD PTR 42

 POP EAX

ESP: 00FF5636

EAX: 52045204

EDX: 0000F36B

00

00

42

00

00

00

64

5F

← ESP

POP/PUSH • OPERACJE NA STOSIE

PUSH EAX

POP DX

POP AX

PUSH DWORD PTR 42

 POP EAX

ESP: 00FF563A

EAX: 00000042

EDX: 0000F36B

00

00

42

00

00

00

64

5F

ESP

CBW/CWDE/CWD · CONVERT BYTE/WORD/?

CBW: AX = AL (BYTE → WORD)

CWDE: EAX = AX (WORD → DWORD)

CWD: DX:AX = AX (WORD → DWORD)

^ OPERACJE ZE ZNAKIEM ^

AX = -123 → CWDE → EAX = -123

CBW/CWDE/CWD • CONVERT BYTE/WORD/?

BEZ ZNAKU?

AND EAX, 0000FFFFh

WYZERUJE WSZYSTKO NAD AX

CBW/CWDE/CWD • CONVERT BYTE/WORD/?

BEZ ZNAKU?

AND EAX, 000000FFh

WYZERUJE WSZYSTKO NAD AL

ADD/ADC/SUB/SBB • ADD/SUBTRACT

A += B		ADD A, B
A += B + CF		ADC A, B
A -= B		SUB A, B
A -= B + CF		SBB A, B
A += 1		INC A
A -= 1		DEC A

ADD/ADC/SUB/SBB • ADD/SUBTRACT

EAX = FFFFFFFF

ADD EAX, 1

EAX = ?????????

ADD/ADC/SUB/SBB • ADD/SUBTRACT

EAX = FFFFFFFF

ADD EAX, 1

EAX = 00000000

ADD/ADC/SUB/SBB • ADD/SUBTRACT

EAX = FFFFFFFF

ADD EAX, 1

EAX = 00000000

CF = 1

ADD/ADC/SUB/SBB • ADD/SUBTRACT

EAX = 00000000

SUB EAX, 1

EAX = FFFFFFFF

ADD/ADC/SUB/SBB • ADD/SUBTRACT

EAX = 00000000

SUB EAX, 1

EAX = FFFFFFFF

GANDHI APPROVES



DIV/MUL • DIVIDE/MULTIPLY

DIV A: $AL = AL / A,$ $AH = \text{RESZTA}$
 $AX = DX:AX / A,$ DX
 $EAX = EDX:EAX / A,$ EDX

MUL A: $AX = AL * A$
 $DX:AX = AX * A$
 $EDX:EAX = EAX * A$

DIV/MUL • DIVIDE/MULTIPLY

DIV A: $AL = AL/A,$ $AH = \text{RESZTA}$

$AX = DX:AX / A,$ DX

$EAX = EDX:EAX / A,$ EDX

$EAX = 360 / 40:$ $MOV\ EDX,\ 0$

$MOV\ EAX,\ 360$

$DIV\ 40$

BT/BTS/BTR/BTC • BIT TESTS

BT A, BIT: CF = BIT NA WYBRANEJ POZYCJI W A

BTS A, BIT: JW, PO CZYM USTAWIA GO NA 1

BTR A, BIT: JW, PO CZYM USTAWIA GO NA 0

BTC A, BIT: JW, PO CZYM USTAWIA GO NA !CF

BT/BTS/BTR/BTC • BIT TESTS

BT A, BIT: CF = BIT NA WYBRANEJ POZYCJI W A

BTS A, BIT: JW, PO CZYM USTAWIA GO NA 1

BTR A, BIT: JW, PO CZYM USTAWIA GO NA 0

BTC A, BIT: JW, PO CZYM USTAWIA GO NA !CF

EAX: 00F37F5Ah -> BT EAX, 4 -> CF = ?

BT/BTS/BTR/BTC • BIT TESTS

BT A, BIT: CF = BIT NA WYBRANEJ POZYCJI W A

BTS A, BIT: JW, PO CZYM USTAWIA GO NA 1

BTR A, BIT: JW, PO CZYM USTAWIA GO NA 0

BTC A, BIT: JW, PO CZYM USTAWIA GO NA !CF

EAX: 00F37F5Ah -> BT EAX, 4 -> CF = ?

5A: 010**1**1010b -----> CF = 1

SAL/SAR/SHL/SHR • BIT SHIFT

AX: 0100 1011 0010 1010b

SHL/SHR: SHIFT LEFT/RIGHT

SAL: TO SAMO CO SHL

SAR: UZUPEŁNIA OSTATNIM BITEM!

SAL/SAR/SHL/SHR • BIT SHIFT

AX: 0100 1011 0010 1010b SHR AX, 4

SHR AX, 4

SAL/SAR/SHL/SHR • BIT SHIFT

AX: 0010 0101 1001 0101b SHR AX, 4

SHR AX, 3

SAL/SAR/SHL/SHR • BIT SHIFT

AX: 0001 0010 1100 1010b SHR AX, 4

SHR AX, 2

SAL/SAR/SHL/SHR • BIT SHIFT

AX: 0000 1001 0110 0101b SHR AX, 4

SHR AX, 1

SAL/SAR/SHL/SHR • BIT SHIFT

AX: 0000 0100 1011 0010b SHR AX, 4

SHR AX, 0 → DONE!

SAL/SAR/SHL/SHR • BIT SHIFT

AX: 0100 1011 0010 1010b SAR AX, 4

SAR AX, 4 → AX JEST DODATNI == SHR

SAL/SAR/SHL/SHR • BIT SHIFT

AX: 1100 1011 0010 1010b SAR AX, 4

SAR AX, 4 → AX JEST UJEMNY

SAL/SAR/SHL/SHR • BIT SHIFT

AX: 1110 0101 1001 0101b SAR AX, 4

SAR AX, 3 → AX JEST UJEMNY

SAL/SAR/SHL/SHR • BIT SHIFT

AX: 1111 0010 1100 1010b SAR AX, 4

SAR AX, 2 → AX JEST UJEMNY

SAL/SAR/SHL/SHR • BIT SHIFT

AX: 1111 1001 0110 0101b SAR AX, 4

SAR AX, 1 → AX JEST UJEMNY

SAL/SAR/SHL/SHR • BIT SHIFT

AX: 1111 1100 1011 0010b SAR AX, 4

SAR AX, 0 → DONE!

ROL/ROR/RCL/RCR • BIT ROTATE

AX: 0100 1011 0010 1010b

ROL/ROR: ROTATE LEFT/RIGHT

RCL/RCR: ROTATE WITH CARRY L/R

ROL/ROR/RCL/RCR • BIT ROTATE

AX: 0100 1011 0010 1010b ROL AX, 4

ROL AX, 4

CF = ?

ROL/ROR/RCL/RCR • BIT ROTATE

AX: 1001 0110 0101 0100b ROL AX, 4

ROL AX, 3

CF = 0

ROL/ROR/RCL/RCR • BIT ROTATE

AX: 0010 1100 1010 1001b ROL AX, 4

ROL AX, 2

CF = 1

ROL/ROR/RCL/RCR • BIT ROTATE

AX: 0101 1001 0101 0010b ROL AX, 4

ROL AX, 1

CF = 0

ROL/ROR/RCL/RCR · BIT ROTATE

AX: 1011 0010 1010 0100b ROL AX, 4

ROL AX, 0 → DONE!

CF = 0

ROL/ROR/RCL/RCR • BIT ROTATE

AX: ?0100 1011 0010 1010b RCL AX, 4

ROL/ROR/RCL/RCR • BIT ROTATE

AX: ?0100 1011 0010 1010b RCL AX, 4

^ WAT

ROL/ROR/RCL/RCR • BIT ROTATE

AX: ?0100 1011 0010 1010b RCL AX, 4

^ CF = ?

ROL/ROR/RCL/RCR · BIT ROTATE

AX: ?0100 1011 0010 1010b

^ CF = ?

RCL AX, 4

RCL AX, 4

ROL/ROR/RCL/RCR · BIT ROTATE

AX: 01001 0110 0101 010?b RCL AX, 4

^ CF = 0 RCL AX, 3

ROL/ROR/RCL/RCR • BIT ROTATE

AX: 10010 1100 1010 10?0b RCL AX, 4

^ CF = 1

RCL AX, 2

ROL/ROR/RCL/RCR · BIT ROTATE

AX: 00101 1001 0101 0?01b RCL AX, 4

^ CF = 0

RCL AX, 1

ROL/ROR/RCL/RCR • BIT ROTATE

AX: 01011 0010 1010 ?010b

^ CF = 0

RCL AX, 4

RCL AX, 0

DONE!

NEG/NOT/AND/OR/XOR · BIT OPS

AND: $1 \& 1 = 1$, ELSE 0

OR: $0 \mid 0 = 0$, ELSE 1

XOR: $1 * 0 = 1$, $0 * 1 = 1$, ELSE 0

NOT: $1 \rightarrow 0$, $0 \rightarrow 1$

NEG/NOT/AND/OR/XOR • BIT OPS

AX: 0010 1100 1010 1001b

AND AX, 0000 1111 1111 0000b

NEG/NOT/AND/OR/XOR • BIT OPS

AX: 0010 1100 1010 1001b

AND AX, 0000 1111 1111 0000b

AX: 0000 1100 1010 0000b

NEG/NOT/AND/OR/XOR • BIT OPS

AX: 0010 1100 1010 1001b

OR AX, 0000 1111 1111 0000b

NEG/NOT/AND/OR/XOR • BIT OPS

AX: 0010 1100 1010 1001b

OR AX, 0000 1111 1111 0000b

AX: 0010 1111 1111 1001b

NEG/NOT/AND/OR/XOR • BIT OPS

AX: 0010 1100 1010 1001b

XOR AX, 0000 1111 1111 0000b

NEG/NOT/AND/OR/XOR • BIT OPS

AX: 0010 1100 1010 1001b

XOR AX, 0000 1111 1111 0000b

AX: 0010 0011 0101 1001b

NEG/NOT/AND/OR/XOR • BIT OPS

AX: 0010 1100 1010 1001b

XOR AX, 0000 1111 1111 0000b

^ WYBIÓRCZA 0011 0101 NEGACJA

NEG/NOT/AND/OR/XOR • BIT OPS

AX: 0010 1100 1010 1001b

NOT AX

AX: 1101 0011 0101 0110b

NEG/NOT/AND/OR/XOR • BIT OPS

AX: 0010 1100 1010 1001b

NEG AX

AX: ?

NEG/NOT/AND/OR/XOR • BIT OPS

AX: 0010 1100 1010 1001b

NEG AX = NOT A, INC A

AX: 1101 0011 0101 0110b

NEG/NOT/AND/OR/XOR • BIT OPS

AX: 0010 1100 1010 1001b

NEG AX = NOT A, INC A

AX: 1101 0011 0101 0111b

NEG/NOT/AND/OR/XOR • BIT OPS

NOT: $A = !A$ | NOT 0101b \rightarrow 1010b

NEG: $A = -A$ | NEG 13 \rightarrow -13

KOPROCESOR

WSZYSTKIE WCZESNIEJSZE INSTRUKCJE
ARYTMETYCZNE PRACUJĄ NA LICZBACH
CAŁKOWITYCH – FLOATY OBSŁUGUJE
KOPROCESOR. ZANIM UŻYJESZ, ODPAL
FINIT ABY GO ZAINICJALIZOWAĆ.

KOPROCESOR

KOPROCESOR MA SWÓJ 8-ELEMENTOWY STOS OD $ST(0)$ DO $ST(7)$ – JEŚLI COŚ TU ŁADUJEMY, WSZYSTKIE ELEMENTY SĄ PRZESUNIĘTE W DÓŁ O 1 POZYCJĘ I $ST(0)$ = ŁADOWANY ELEMENT.

CO MOŻNA ZAŁADOWAĆ NA STOS X87

FLD A: FLOAT Z ST(0..6)/PAMIĘCI

FILD A: INTEGER Z PAMIĘCI

FLDZ: 0, **FLD1:** 1 **FLDPI:** LICZBA PI

FLDL2E: $\text{LOG}_2(E)$ **FLDL2T:** $\text{LOG}_2(10)$

FLDLG2: $\text{LOG}_{10}(2)$ **FLDLN2:** $\text{LN}(2)$

OPERACJE X87

FST: SKOPIUJ $ST(0)$ DO $ST(*)$ /PAMIĘCI

FSTP: JAK WYŻEJ, PLUS USUWA $ST(0)$

FIST(P): ZAOKRĄGL DO SIGNED INTA I ZAPISZ DO PAMIĘCI (PO CZYM USUŃ $ST(0)$)

FXCH A: ZAMIEŃ $ST(0)$ Z PODANYM $ST(*)$

FCHS: ZMIEŃ ZNAK $ST(0)$ NA PRZECIWNY

OPERACJE X87

F??? A: ST(0) <?> A (PAMIĘĆ Z FLOATEM)

F??? A, B: A <?> B, ST(0) MUSI BYĆ A/B

F???P: ST(1) <?> ST(0), POP ST(0)

F???P A, ST(0): A <?> ST(0), POP ST(0)

FI??? A: ST(0) <?> DOUBLE_CAST(A)

OPERACJE: ADD, SUB, MUL, DIV

OPERACJE X87

POZOSTAŁE OPERACJE NA ST(0):

FSQRT: PIERWIASTEK KWADRATOWY

FSIN/FCOS: SINUS/COSINUS

FABS: WARTOŚĆ CAŁKOWITA

F2XM1: $(2 * ST(0)) - 1$

FRNDINT: ZAOKRĄGL DO NAJBLIŻSZEGO INTA

OPERACJE X87

CO ZWYKLE ROBIMY Z KOPROCESOREM:

- ŁADUJEMY DANE NA STOS (FILD, FLD, ...)
- WYKONUJEMY OBLICZENIA WG ZADANIA
- ZRZUCAMY DO PAMIĘCI
- ODCZYTUJEMY Z PAMIĘCI DO REJESTRU

PORÓWNANIA I SKOKI

TO WSZYSTKO WYKONUJE SIĘ LINIOWO,
JEDNO PO DRUGIM. JAK KONTROLOWAĆ
PRZEPŁYW PROGRAMU?

DZIAŁANIA → PORÓWNANIA → SKOK

PORÓWNANIA I SKOKI

CMP **A**, **B** – COMPARE (ODEJMIJ I USTAW
FLAGI – DZIAŁA JAK **SUB**)

FCOMI **ST(0)**, **B** – COMPARE
(KOPROCESOR, B = ST(*))

JMP ADRES – SKOK BEZWARUNKOWY

```
LOOP_DA_LOOP:
```

```
; SOME OPS
```

```
JMP LOOP_DA_LOOP
```

JMP ADRES – SKOK BEZWARUNKOWY

```
LOOP_DA_LOOP:      DO {  
    ; SOME OPS      // SOME OPS  
    JMP LOOP_DA_LOOP } WHILE (TRUE);
```

PORÓWNIANIA I SKOKI

SKOKI WARUNKOWE:

J [N] [A/B] [E]

J [N] [G/L] [E]

J [N] [C/O/S/Z]

PORÓWNANIA I SKOKI

SKOKI WARUNKOWE:

JUMP IF [NOT] [ABOVE/BELOW] [OR EQUAL]

JUMP IF [NOT] [GREATER/LOWER] [OR EQUAL]

JUMP IF [NOT] [CARRY/OVERFLOW/SIGN/ZERO]

PORÓWNANIA I SKOKI

SKOKI WARUNKOWE:

BEZ ZNAKU:	[ABOVE/BELOW]	[OR EQUAL]
ZE ZNAKIEM:	[GREATER/LOWER]	[OR EQUAL]
FLAGI:	[CARRY/OVERFLOW/SIGN/ZERO]	

SKOK WARUNKOWY

DO_WHILE:

; SOME OPS

CMP ???, ???

J??? DO_WHILE

DO {

// SOME OPS

} **WHILE** (???);

PORÓWNANIA I SKOKI

JAKA INSTRUKCJA DLA WARUNKU?

WHILE (ECX <= 32), ECX = RANGE(0..31)

J [N] [A/B] [E]

J [N] [G/L] [E]

J [N] [C/O/S/Z]

PORÓWNANIA I SKOKI

JAKA INSTRUKCJA DLA WARUNKU?

WHILE (ECX <= 32), ECX = RANGE(0..31)

CMP ECX, 32

JBE DO_WHILE

J [N] [A/B] [E]

J [N] [G/L] [E]

J [N] [C/O/S/Z]

PORÓWNANIA I SKOKI

JAKA INSTRUKCJA DLA WARUNKU?

WHILE (AL > 0), AL = BAJT -128..127

J [N] [A/B] [E]

J [N] [G/L] [E]

J [N] [C/O/S/Z]

PORÓWNANIA I SKOKI

JAKA INSTRUKCJA DLA WARUNKU?

WHILE (AL > 0), AL = BAJT -128..127

CMP AL, 0

JG DO_WHILE

J [N] [A/B] [E]

J [N] [G/L] [E]

J [N] [C/O/S/Z]

DO WHILE (AL > 0 && ECX < 64)

DO_WHILE:

; DO STUFF

CMP AL, 0 ; AL: -128..127

JNG END_DO_WHILE

CMP ECX, 64 ; ECX: 0..64

JB DO_WHILE

END_DO_WHILE:

; ETC...

CMP DZIAŁA RAZ –
KILKA JMP POD RZĄD?

J [N] [A/B] [E]

J [N] [G/L] [E]

J [N] [C/O/S/Z]

DO WHILE (AL > 0 && ECX < 64)

DO_WHILE:

; DO STUFF

CMP AL, 0 ; AL: -128..127

JNG END_DO_WHILE

CMP ECX, 64 ; ECX: 0..64

JB DO_WHILE

END_DO_WHILE:

; ETC...

CMP DZIAŁA RAZ –
KILKA JMP POD RZĄD?

J [N] [A/B] [E]

J [N] [G/L] [E]

J [N] [C/O/S/Z]

DO WHILE (AL > 0 || ECX < 64)

DO_WHILE:

; DO STUFF

CMP AL, 0 ; AL: -128..127

JG DO_WHILE

CMP ECX, 64 ; ECX: 0..64

JB DO_WHILE

END_DO_WHILE:

; ETC...

CMP DZIAŁA RAZ –
KILKA JMP POD RZĄD?

J [N] [A/B] [E]

J [N] [G/L] [E]

J [N] [C/O/S/Z]

DO WHILE (AL > 0 || ECX < 64)

DO_WHILE:

; DO STUFF

CMP AL, 0 ; AL: -128..127

JG DO_WHILE

CMP ECX, 64 ; ECX: 0..64

JB DO_WHILE

END_DO_WHILE: ; ← PO CO TO?

; ETC...

**CMP DZIAŁA RAZ –
KILKA JMP POD RZĄD?**

J [N] [A/B] [E]

J [N] [G/L] [E]

J [N] [C/O/S/Z]

DO WHILE (AL > 0 || ECX < 64)

DO_WHILE:

; DO STUFF

CMP AL, 0 ; AL: -128..127

JG DO_WHILE

CMP ECX, 64 ; ECX: 0..64

JB DO_WHILE

END_DO_WHILE: ; ← PO CO TO?

; ETC...

BREAK;

V

JMP END_DO_WHILE

IF (...) { ... }

CMP ???, ???

JN?? END_IF_STUFF

IF_STUFF:

; SOME OPS

END_IF_STUFF:

; OPS AFTER IF

IF (???) {

// SOME OPS

}

IF (...) { ... }

```
CMP [ESI], 4
JNL END_IF_STUFF
IF_STUFF:
    ; SOME OPS
END_IF_STUFF:
    ; OPS AFTER IF
```

```
IF ([ESI] < 4) {
    // SOME OPS
}
```

IF (...) { ... } ELSE (...) { ... }

```
CMP [ESI], 4
JNL ELSE_STUFF
IF_STUFF:
    ; SOME OPS
    JMP END_IF_STUFF
ELSE_STUFF:
    ; OTHER OPS
END_IF_STUFF:
    ; OPS AFTER IF
```

```
IF ([ESI] < 4) {
    // SOME OPS
} ELSE {
    // OTHER OPS
}
```

WHILE (...) { ... }

WHILE_STUFF:

CMP EDI, 0

JNE END_IF_STUFF

; SOME OPS

END_WHILE_STUFF:

; OPS AFTER IF

WHILE (EDI == 0) {

// SOME OPS

}

WHILE (...) { ... }

WHILE_STUFF:

CMP EDI, 0

JNZ END_IF_STUFF

; SOME OPS

END_WHILE_STUFF:

; OPS AFTER IF

WHILE (EDI == 0) {

// SOME OPS

}

```
FOR (...) { ... }
```

```
FOR (INT I = 0; I < 32; I++) { ... }
```

```
FOR (...) { ... }
```

```
FOR (ECX = 0; ECX < 32; ECX++) { ... }
```

```
FOR (ECX = 0; ECX < 32; ECX++) {...}
```

```
ECX = 0;
```



```
DO {
```

```
// SOME OPS
```

```
ECX++;
```

```
} WHILE (ECX < 32);
```

FOR (ECX = 0; ECX < 32; ECX++) {...}

```
ECX = 0;  MOV ECX, 0
DO {
    // SOME OPS
    ECX++;  INC ECX
} WHILE (ECX < 32);    CMP ???, ???
                        J??? DO_WHILE
                        END_DO_WHILE:
                        ; ETC...
```

ZADANKO?

KOLOKWIUM 8.12.2016, 6 PUNKTÓW

TREŚĆ: [HTTP://AKO.DRAGONIC.EU](http://ako.dragonic.eu)

ZADANKO!

NAPISZ PROGRAM W 32-BITOWYM ASMIE
OBLICZAJĄCY PROSTE ŚREDNIE KROCZĄCE DLA
PEWNEGO SZEREGU CZASOWEGO (SMA). JEST
TO ŚREDNIA ARYTMETYCZNA Z **N** OSTATNICH
POMIARÓW. DODATKOWYM PARAMETREM JEST
PRZESUNIĘCIE POCZĄTKU OKNA O **K**
KOLEJNYCH ELEMENTÓW SZEREGU PRZY
WYZNACZANIU KOLEJNEJ ŚREDNIEJ.

ZADANKO!

P = {P₀, P₁, P₂, ...}, **N** = 4, **K** = 2

{P₀, P₁, P₂, P₃}, {P₂, P₃, P₄, P₅}, ...

ELEMENTY SZEREGU MAJĄ PO 16 BITÓW, WARTOŚĆ N ZNAJDUJE SIĘ W ECX, WARTOŚĆ K W EBX ($0 < K \leq N$), ESI WSKAZUJE POŁOŻENIE PIERWSZEGO ELEMENTU, EDX LICZBĘ WSZYSTKICH ELEMENTÓW SZEREGU ($M > N$). W REJESTRZE EDI ZNAJDUJE SIĘ ADRES PAMIĘCI DO KTÓREGO NALEŻY WPISYWAĆ KOLEJNE 16-BITOWE ŚREDNIE SMA DLA ZADANYCH PARAMETRÓW.

ZADANKO!

$P = \{P_0, P_1, P_2, P_3, P_4, P_5, P_6, \dots\}$
 $\{P_0, P_1, P_2, P_3\}, \{P_2, P_3, P_4, P_5\}, \dots$

DLA $N = 4$, $K = 2$, ZACZYNAMY OD P_0 I SUMUJEMY NASTĘPNE 4 ELEMENTY ($P_0..P_3$), PO CZYM DZIELIMY PRZEZ 4. POTEM SKACZEMY O 2 ELEMENTY DO PRZODU (DO P_2), ZNOWU SUMUJEMY 4 ELEMENTY ($P_2..P_5$) I DZIELIMY PRZEZ 4. I TAK DALEJ, AŻ POLICZYMY ŚREDNIE DLA WSZYSTKICH TAKICH PODCIĄGÓW Z DANEGO NAM CIĄGU P (ZNAMY JEGO DŁUGOŚĆ).

ZADANKO!

→ **CODE**

KONIEC

DZIĘKI!

**YOUR HUMBLE HOST
RYSIEK KNOP**

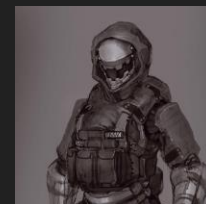


FEEDBACK PLS?

[HTTP://AKO.DRAGONIC.EU](http://ako.dragonic.eu)

KONIEC

YOUR HUMBLE HOST
RYSIEK KNOP



SLAJDY I MATERIAŁY DOSTĘPNE NA LICENCJI
CREATIVE COMMONS ATTRIBUTION 4.0 INTERNATIONAL



UŻYTO ZADAŃ NA KOLOKWIUM Z AKO
DR INŻ TOMASZA DZIUBICHA

