ReverseCraft

assembler

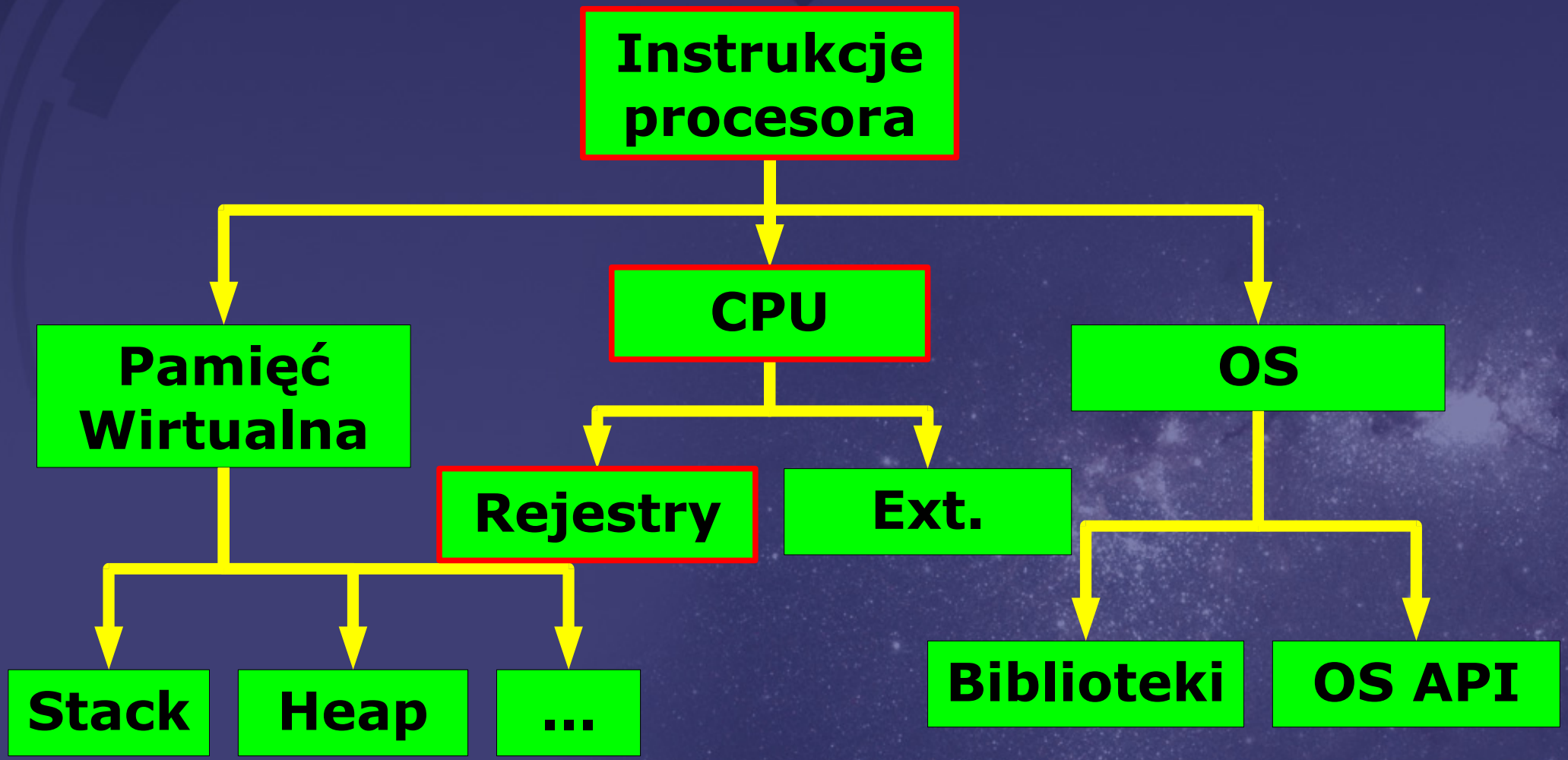
by gynvael.coldwind//vx

002 – Opcode

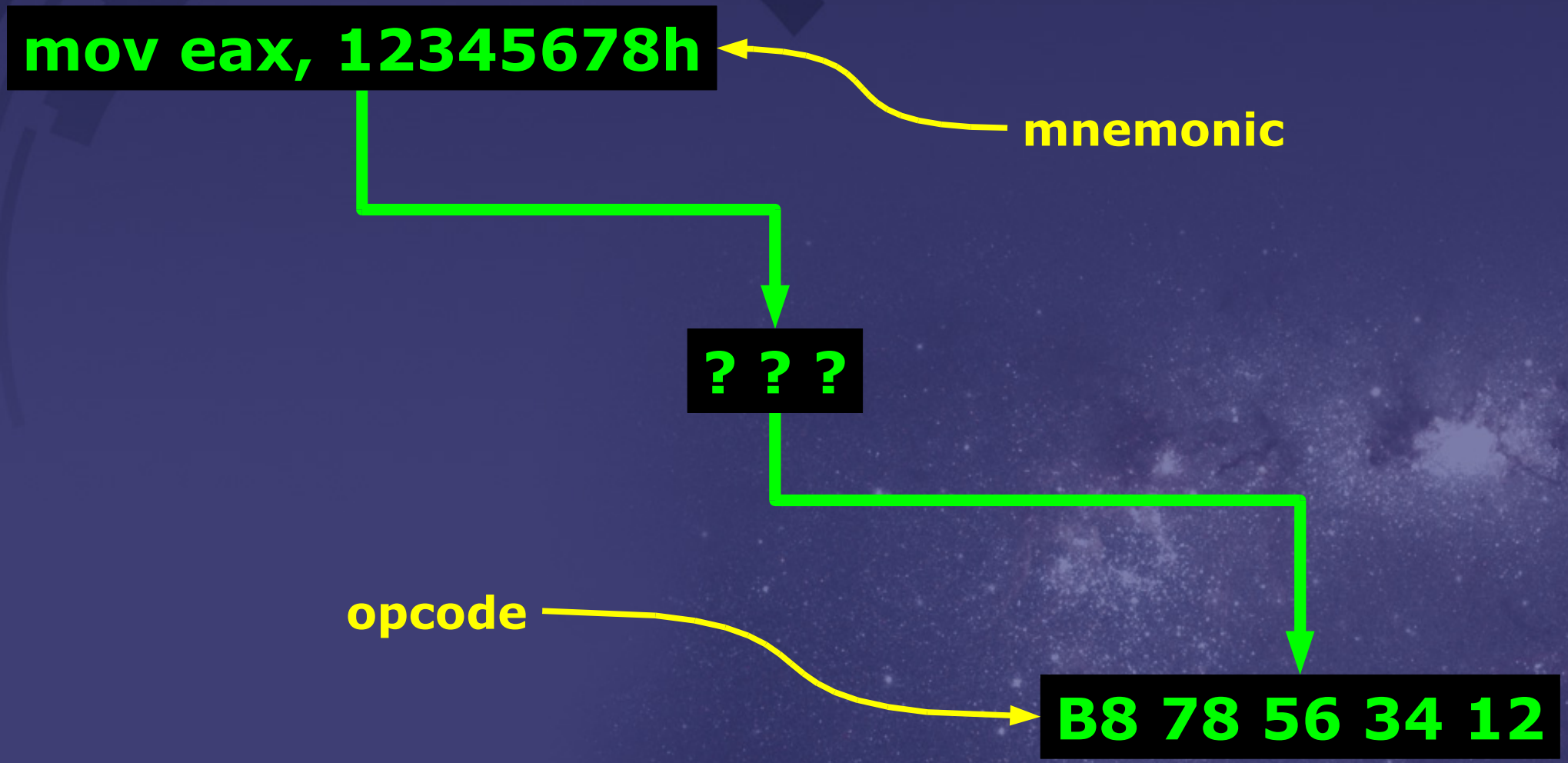
Strony projektu:

http://re.coldwind.pl/
http://www.uw-team.org/

Zasoby!
czyli co możemy użyć...



Instrukcje procesora



Instrukcje procesora

Dla wszystkich:

- opcode'y, prefixy, imm
- co można dać w [ ]
- jak czytać opcodes.txt :)

Dla dociekliwych:

- jak dokładnie zbudowany jest opcode
- co zawiera ModRM i SIB
- jak wygląda mapa opcode'ów

Instrukcje procesora

(opcodes.txt/pentium.txt)

Instrukcje procesora

(opcodes.txt/pentium.txt)

/r -- Indicates that the ModR/M byte of the instruction contains both a register operand and an r/m operand.

???args???

opcode

mnemonic

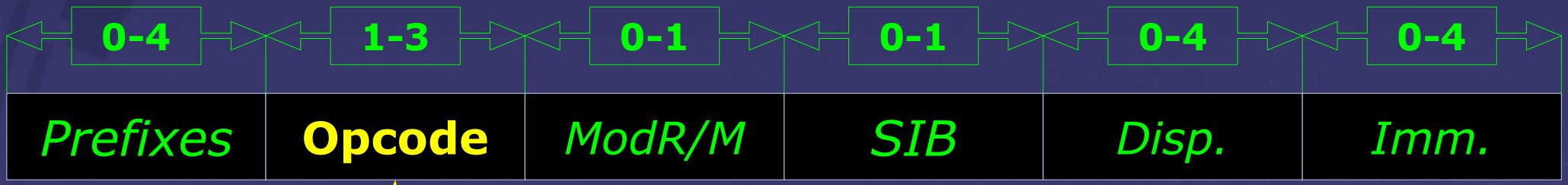
opis

Instrukcje procesora

(Intel® Manual, Vol 2a, rozdział 2.1)

Instrukcje procesora

(Intel® Manual, Vol 2a, rozdział 2.1)



unikatowy identyfikator polecenia procesora

Przykłady poleceń:

```
60 - PUSHAD
90 - NOP
90 - XCHG EAX, EAX
61 - POPAD
C3 - RET
D9 FE - FSIN
```

Jak wyglądają opcode'y
(Intel® Manual, Vol 2b, dodatek A i B)

Uproszczona mapa opcode'ów

00-FF oprócz poniższych wyjątków, to pojedyncze instrukcje, prefiksy oraz grupy instrukcji z indexem w MODR/M

Wyjątki

D8-DF to „ucieczka” do instrukcji koprocesora (FPU)
0F to „ucieczka” do rozszerzonego zestawu instrukcji, podobnie jak sekwencje:
66 0F
F2 0F
F3 0F
66 0F

Jak wyglądają opcode'y
(Intel® Manual, Vol 2b, dodatek A i B)

```
C3 - RET

40+rd - INC rejestr
rd = (EAX = 0, ECX, EDX, EBX, ESP, EBP, ESI, EDI)
40 - INC EAX
41 - INC EDX
47 - INC EDI

C0 - grupa opcode'ów: ???
C0 /0 ib - ROL r/m8, imm8
C0 /1 ib - ROR r/m8, imm8
...
C0 /7 ib - SAR r/m8, imm8
```

Jak wyglądają opcode'y
(Intel® Manual, Vol 2b, dodatek A i B)

```
D8 numer_instrukcji_koprocesora lub MODR/M
D8 /0 - FADD m32real
D8 D1 - FCOM
D8 D9 - FCOMP

Escape prefix 0F
0F BE /r - MOVSX r32,r/m8
0F BF /r - MOVSX r32,r/m16
(B8+rd - MOV r32,imm32)

66 0F 3A 0F - PALIGNR xmm1, xmm2/m128, imm8
```

Jak wyglądają opcode'y
(Intel® Manual, Vol 2b, dodatek A i B)

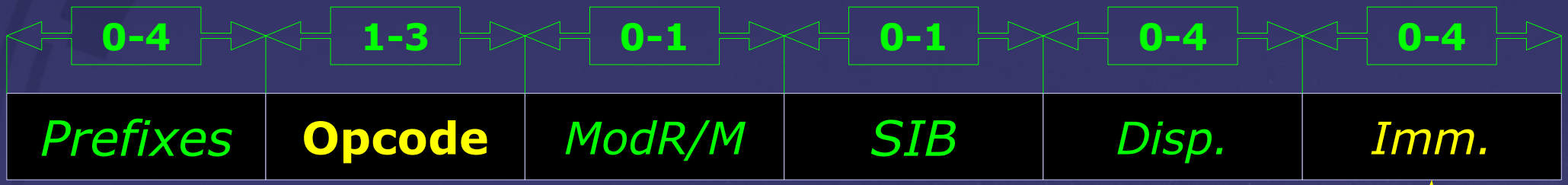
```
D8 numer_instrukcji_koprocesora lub MODR/M
D8 /0 - FADD m32real
D8 D1 - FCOM
D8 D9 - FCOMP

Escape prefix 0F
0F BE /r - MOVSX r32,r/m8
0F BF /r - MOVSX r32,r/m16
(B8+rd - MOV r32,imm32)

66 0F 3A 0F - PALIGNR xmm1, xmm2/m128, imm8
```

Jak wyglądają opcode'y
(Intel® Manual, Vol 2b, dodatek A i B)

```
// 60 - PUSHAD, 90 - NOP, 61 - POPAD, C3 - RET

#include <stdio.h>

int main(void)
{
  char *func_code = „\x60\x90\x61\xC3”;
  void (*func_ptr)(void) = (void(*)(void))func_code;

  func_ptr();

  return 0;
}
```

Instrukcje procesora

(Intel® Manual, Vol 2a, rozdział 2.1)



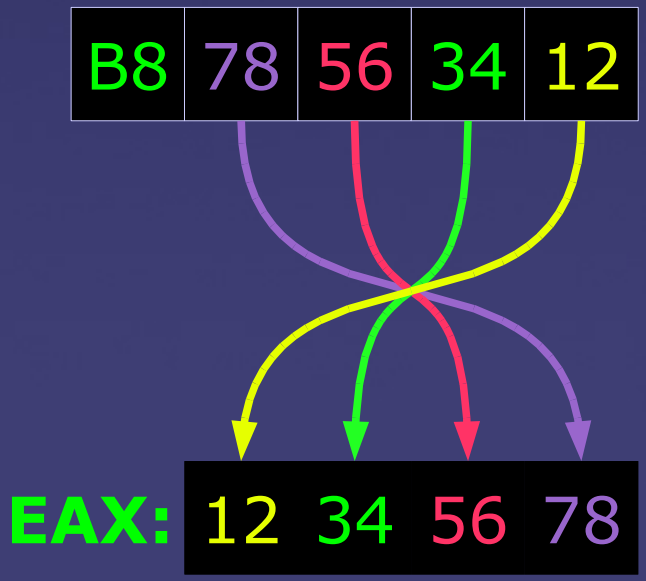
Przykłady poleceń:

```
B8+rd - MOV r32,imm32
        np.
   B8 78 56 34 12
        to
MOV EAX, 0x12345678

!LITTLE ENDIAN!
```

stała (liczba) / natychmiastowa / bezpośrednia (immediate)

Little big endian

Little endian (X86)



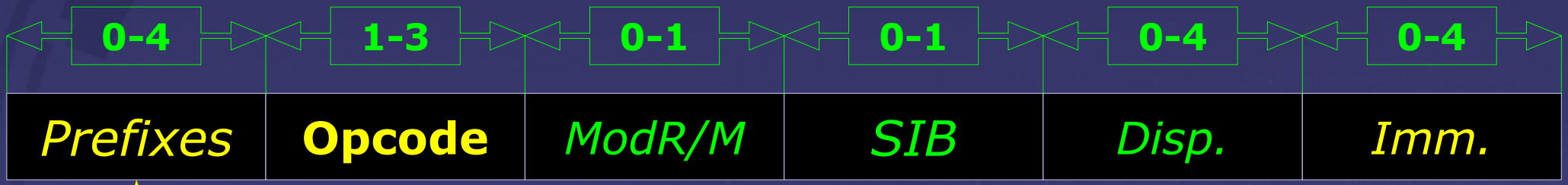
Big endian (SPARC)

Program cd...

```
// B8+rd imm32 - MOV rd, imm32  ;  C3 - RET

#include <stdio.h>

int main(void)
{
  char *func_code = "\xB8\x78\x56\x34\x12\xC3";
  int (*func_ptr)(void) = (int(*)(void))func_code;

  printf("%x\n", func_ptr());

  return 0;
}
```

Instrukcje procesora

(Intel® Manual, Vol 2a, rozdział 2.1)



prefiksy zmieniające działanie instrukcji

4 grupy prefixów:

1. Lock and repeat
2. Segment override / branch hint
3. Operand-size override
4. Address-size override

Instrukcje procesora

4 grupy prefixów:

1. Lock and repeat
F0 - LOCK, F2 - REPNE/REPNZ, F3 - REP or REPE/REPZ

2. Segment override / branch hint
2E - CS, 36 - SS, 3E - DS, 26 - ES, 64 - FS, 65 - GS
2E - branch not taken, 3E - branch taken

3. Operand-size override (and espace)
66 - Operand-size override prefix

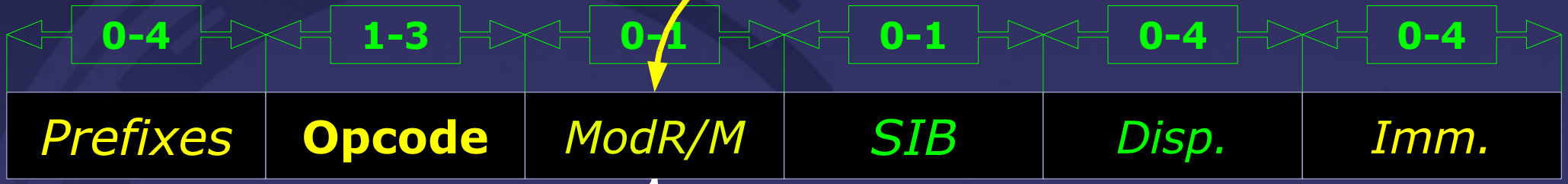
4. Address-size override
67 - Address-size override prefix

Instrukcje procesora

4 grupy prefixów:

```
66 - Operand-size override prefix
operand 32 bity <-> operand 16 bitów

   B8 78 56 34 12 - MOV EAX, 0x12345678
66 B8 34 12       - MOV AX, 0x1234

rejestr 32-bitowy (EAX) -> rejestr 16-bitowy (AX)
stała 32-bitowa         -> stała 16-bitowa
```

prefiksy zmieniające działanie instrukcji



	Mod	Reg / Op	R/M	
0	[REG]	EAX	EAX	0
1	[REG+disp8]	ECX	ECX	1
2	[REG+disp32]	EDX	EDX	2
3	REG	EBX	EBX	3
		ESP	SIB/ESP	4
		EBP	disp32/EBP	5
		ESI	ESI	6
		EDI	EDI	7

```
C7 /0 id MOV r/m32,imm32
Opcode = C7
ModR/M = ( Mod=?? Op=000 R/M=??? )

C7 C1 11 22 33 44 -> MOV ECX, 0x44332211
   C1 = 11000001 = ( Mod=11 Op=000 R/M=001 )

C7 01 11 22 33 44 -> MOV [ECX], 0x44332211
   01 = 00000001 = ( Mod=00 Op=000 R/M=001 )
```

	2	3	3	
	Mod	**Reg / Op**	**R/M**	
0	[REG]	EAX	EAX	0
1	[REG+disp8]	ECX	ECX	1
2	[REG+disp32]	EDX	EDX	2
3	REG	EBX	EBX	3
		ESP	SIB/ESP	4
		EBP	disp32/EBP	5
		ESI	ESI	6
		EDI	EDI	7

```
8B     /r       MOV r32,r/m32
89     /r       MOV r/m32,r32

8B 00 -> MOV EAX, [EAX]
89 00 -> MOV [EAX], EAX

   00 = ( Mod=00, Reg=000, R/M=000 )
        ( [REG]      EAX       EAX   )
```

	Mod	Reg / Op	R/M	
0	[REG]	EAX	EAX	0
1	[REG+disp8]	ECX	ECX	1
2	[REG+disp32]	EDX	EDX	2
3	REG	EBX	EBX	3
		ESP	SIB/ESP	4
		EBP	disp32/EBP	5
		ESI	ESI	6
		EDI	EDI	7

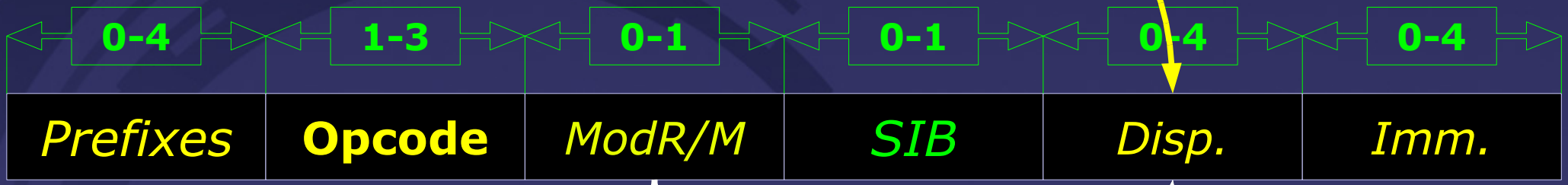
Kodowanie MOV EAX, EAX

```
8B     /r      MOV r32,r/m32
8B C0 -> MOV EAX, EAX

89     /r      MOV r/m32,r32
89 C0 -> MOV EAX, EAX
```

Kodowanie MOV EAX, IMM32

```
B8+rd          MOV r32,imm32
B8 IMM32 -> MOV EAX, IMM32

C7     /0 id   MOV r/m32,imm32
C7 C0 IMM32 -> MOV EAX, IMM32
```

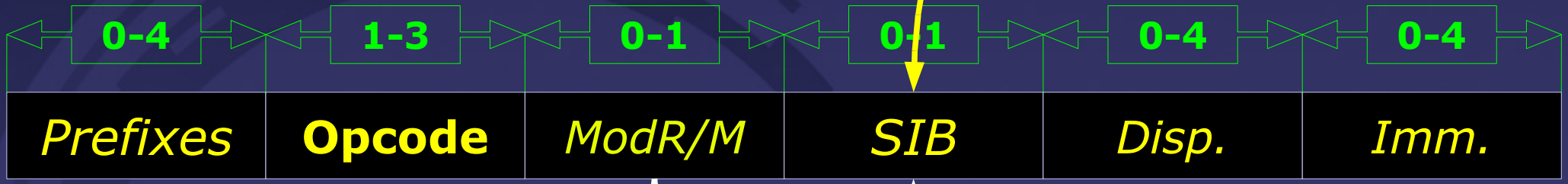
offset, przesunięcie



	Mod	Reg / Op	R/M	
0	[REG]	EAX	EAX	0
1	[REG+disp8]	ECX	ECX	1
2	[REG+disp32]	EDX	EDX	2
3	REG	EBX	EBX	3
		ESP	SIB/ESP	4
		EBP	disp32/EBP	5
		ESI	ESI	6
		EDI	EDI	7

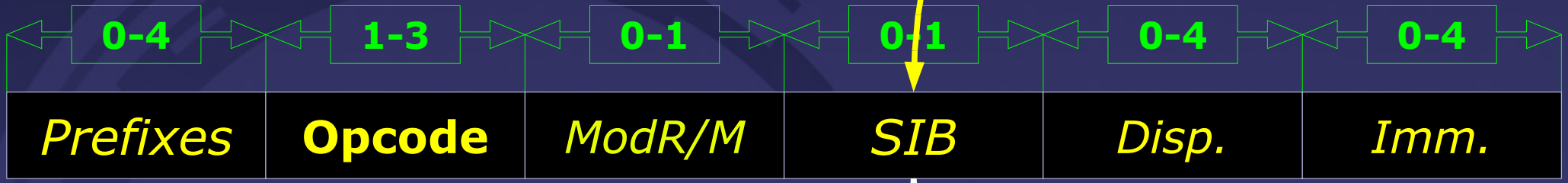
```
MOV EAX, [EAX+0x12345678]
8B 80 78 56 34 12
Mod=10 Reg=000 R/M=000

MOV EAX, [0x12345678]
8B 05 78 56 34 12

MOV EAX, [0x12345678]
A1 78 56 34 12
```

rozszerzona notacja adresu (tablice!)



	Mod	Reg / Op	R/M	
0	[REG]	EAX	EAX	0
1	[REG+disp8]	ECX	ECX	1
2	[REG+disp32]	EDX	EDX	2
3	REG	EBX	EBX	3
		ESP	SIB/ESP	4
		EBP	DISP32/EBP	5
		ESI	ESI	6
		EDI	EDI	7

rozszerzona notacja adresu (tablice!)



	Mod
0	[Base + Index*Scale]
1	[Base + Index*Scale + disp8]
2	[Base + Index*Scale + disp32]
3	REG

	Scale	Index	Base	
0	$2^0 = 1$	EAX	EAX	0
1	$2^1 = 2$	ECX	ECX	1
2	$2^2 = 4$	EDX	EDX	2
3	$2^3 = 8$	EBX	EBX	3
		none	ESP	4
		EBP	Mod=0 -> disp32 Mod=1/2 -> EBP	5
		ESI	ESI	6
		EDI	EDI	7

```
MOV EAX, [EBX+ECX*4 + 0x12345678]
MOV EAX, [ESP+0]
LEA EAX, [ECX+ECX*8 + 0x1234]
```

	Mod
0	[Base + Index*Scale]
1	[Base + Index*Scale + disp8]
2	[Base + Index*Scale + disp32]
3	REG

	2	3	3	
	Scale	Index	Base	
0	$2^0 = 1$	EAX	EAX	0
1	$2^1 = 2$	ECX	ECX	1
2	$2^2 = 4$	EDX	EDX	2
3	$2^3 = 8$	EBX	EBX	3
		none	ESP	4
		EBP	Mod=0 -> disp32 Mod=1/2 -> EBP	5
		ESI	ESI	6
		EDI	EDI	7

Zabawa w disassembler!

```
8B 84 8B 78 56 34 12

Co to za instrukcja ?
```

	Mod
0	[Base + Index*Scale]
1	[Base + Index*Scale + disp8]
2	[Base + Index*Scale + disp32]
3	REG

	Scale	Index	Base	
				2 3 3
0	$2^0 = 1$	EAX	EAX	0
1	$2^1 = 2$	ECX	ECX	1
2	$2^2 = 4$	EDX	EDX	2
3	$2^3 = 8$	EBX	EBX	3
		none	ESP	4
		EBP	Mod=0 -> disp32 Mod=1/2 -> EBP	5
		ESI	ESI	6
		EDI	EDI	7

Zabawa w disassembler!

```
                    8B 84 8B 78 56 34 12

Opcode = 8B /r MOV r32, r/m32
ModR/M = 84 = (Mod=10     Reg=000 R/M=100)
              [+disp32]     EAX     SIB
SIB = 8B = (Scale=10 Index=001 Base=011)
             2^2=4          ECX     EBX

Końcowa postać: MOV EAX, [EBX + ECX*4 + 0x12345678]
```

Podsumowanie

- pełna instrukcja w kodzie maszynowym ma od 1 do 16 bajtów

- instrukcja dzieli się na 6 pól

Prefixes	**Opcode**	*ModR/M*	*SIB*	*Disp.*	*Imm.*

- jakie warianty danego mnemonika mogą być zakodowane można zobaczyć w opcodes.txt/pentium.txt oraz w manualach

- niektóre instrukcje można zakodować na więcej niż jeden sposób

Podsumowanie

- adres w nawiasie [ ] może być złożony z:
[rejestru]
[rejestru + przesunięcia]
[bazy + indeksu*skala]
[bazy + indeksu*skala + przesunięcia]
[indeksu*skala + przesunięcia]
[przesunięcia]
innych opcji nie ma

- przesunięcie to stała (liczba) 8 lub 32 bitowa

Podsumowanie

- z uwagi na budowę instrukcji, nie jest możliwe zapisanie w instrukcji dwóch dereferencji adresu (tj. dwóch członów w [ ])

- prefiksy m.in. umożliwiają przejście między 32 a 16 bitową wielkością operandu

- opcodes.txt jest proste i przydatne :)

Eksperymenty domowe

- co się stanie jeżeli przed opcode wstawimy dwa prefiksy zmieniające wielkość operandu ?

Co dalej?

- prefix zmieniający tryb adresowania z 32-bitowego na 16-bitowe zmienia działanie bajtu ModR/M, sprawdź jak

- sprawdź jakie są rodzaje innych argumentów jakie przyjmują opcode'y (tj. innych niż r32, r/m32 i imm32)

- jakiego rodzaju argumenty przyjmują skoki ?

Dziękuje za uwagę :)

Strony projektu:

http://re.coldwind.pl/
http://www.uw-team.org/