Adresní režimy

VAX ISA

Souhrn VAX ISA

- 32-bitová architektura CISC
- 16 32-bitových registrů (r0, r1, .., r15)
  - r12, r13, r14, r15 rezervováno pro AP, FP, SP, PC
    (AP=argument pointer, FP=Frame pointer – ukazatele do zásobníku při volání fcí)
- 300+ instrukcí s proměnnou délkou
- 22 adresních režimů
- ISA navržena pro zjednodušení kompilátorů a redukci velikosti programu

Architektura VAX

- DEC 1978 VAX-11/780
- Datové typy

Bits	Data type	MIPS name	VAX name
8	Integer	Byte	Byte
16	Integer	Half word	Word
32	Integer	Word	Long word
32	Floating point	Single precision	F_floating
64	Integer	Double word	Quad word
64	Floating point	Double precision	D_floating or G_floating
8n	Character string	Character	Character

Datové typy

Data Type	Size	Range (decimal)	
Integer		Signed	Unsigned
Byte	8 bits	-128 to +127	0 to 255
Word	16 bits	-32768 to +32767	0 to 65535
Longword	32 bits	-2^{31} to $+2^{31}-1$	0 to $2^{32}-1$
Quadword	64 bits	-2^{63} to $+2^{63}-1$	0 to $2^{64}-1$
Octaword	128 bits	-2^{127} to $+2^{127}-1$	0 to $2^{128}-1$
Float			
F_floating	32 bits	Approx. 7 decimal digits precision.	
D_floating	64 bits	Approx. 16 decimal digits precision.	
G_floating	64 bits	Approx. 15 decimal digits precision.	
H_floating	128 bits	Approx. 33 decimal digits precision.	
Packed Decimal String	0 to 16 bytes (31 digits)	Numeric, two digits per byte, sign in low half of last byte	
Character String	0 to 65535 bytes	One character per byte	
Variable-length Bit Field	0 to 31 bytes (DIGITS)	$-10^{31}-1$ to $+10^{31}-1$	
Queue	>= 2 longwords / queue entry	0 through 2 billion entries	

Recreated from Table 2-1 in VAX Architecture Handbook (1981)

Koncepce *word* u VAXu se liší od koncepce *word* u MIPS.
VAX *Longword* je ekvivalentní některému z 4 bytových *words*.

VAX je 32-bitová architektura, instrukce operují nad různými datovými formáty.
Nejčastější:

- Integer/Float různé délky
- Řetězce znaků

Mnoho „hardwarových“ datových typů pro zjednodušení kompilátoru

Příklady instrukcí (1/3)

Instruction type	Example	Instruction meaning
Data transfers	Move data between byte, half-word, word, or double-word operands; * is data type	
	mov*	Move between two operands
	movzb*	Move a byte to a half word or word, extending it with zeros
	mova*	Move the 32-bit address of an operand; data type is last
	push*	Push operand onto stack
Arithmetic/logical	Operations on integer or logical bytes, half words (16 bits), words (32 bits); * is data type	
	add*_	Add with 2 or 3 operands
	cmp*	Compare and set condition codes
	tst*	Compare to zero and set condition codes
	ash*	Arithmetic shift
	clr*	Clear
	cvtb*	Sign-extend byte to size of data type

Figure E.4 Classes of VAX instructions with examples. The asterisk stands for multiple data types: b, w, l, d, f, g, h, and q. The underline, as in addd_, means there are 2-operand (addd2) and 3-operand (addd3) forms of this instruction.

Příklady instrukcí (2/3)

Control	Conditional and unconditional branches	
	`beql, bneq`	Branch equal, branch not equal
	`bleq, bgeq`	Branch less than or equal, branch greater than or equal
	`brb, brw`	Unconditional branch with an 8-bit or 16-bit address
	`jmp`	Jump using any addressing mode to specify target
	`aobleq`	Add one to operand; branch if result ≤ second operand
	`case_`	Jump based on case selector
Procedure	Call/return from procedure	
	`calls`	Call procedure with arguments on stack (see Section E.6)
	`callg`	Call procedure with FORTRAN-style parameter list
	`jsb`	Jump to subroutine, saving return address (like MIPS `jal`)
	`ret`	Return from procedure call

Figure E.4 Classes of VAX instructions with examples. The asterisk stands for multiple data types: b, w, l, d, f, g, h, and q. The underline, as in `addd_`, means there are 2-operand (`addd2`) and 3-operand (`addd3`) forms of this instruction.

Příklady instrukcí (3/3)

Floating point	Floating-point operations on D, F, G, and H formats	
	addd_	Add double-precision D-format floating numbers
	subd_	Subtract double-precision D-format floating numbers
	mulf_	Multiply single-precision F-format floating point
	polyf	Evaluate a polynomial using table of coefficients in F format
Other	Special operations	
	crc	Calculate cyclic redundancy check
	insque	Insert a queue entry into a queue

Figure E.4 Classes of VAX instructions with examples. The asterisk stands for multiple data types: b, w, l, d, f, g, h, and q. The underline, as in addd_, means there are 2-operand (addd2) and 3-operand (addd3) forms of this instruction.

Varianty instrukcí

- Instrukční varianty
  - Operační kód + datový typ + # operandů
    - Operace – add, sub …
    - Datový typ – byte, word, dword,
    - # operandů – 1 ~ 3 registry a 1~3 paměťové reference (závislé na operaci)
- Jako příklad — operace ADD

The operation add works with data types byte, word, long, float, and double and comes in versions for either 2 or 3 unique operands, so the following instructions are all found in the VAX:

```
addb2    addw2    addl2    addf2    addd2
addb3    addw3    addl3    addf3    addd3
```

Celkový počet kombinací instrukcí je 304+, nezahrneme-li variace adresních režimů!

Adresní režimy

- General Register Addressing
  - Literal (3 types)
  - Register R
  - Register deferred (R)
  - Autodecrement and R++ Autoincrement
  - Autoincrement deferred (R++)
  - Byte, Word, Longword displacement
  - Byte, Longword displacement deferred
  - Indexed
- Program Counter Addressing
  - Immediate
  - Absolute
  - Byte relative
  - Byte relative deferred
  - Word relative
  - Word relative deferred
  - Longword relative
  - Longword relative deferred

Register – Register Register–Memory Memory – Memory

Celkem 22 adresních režimů

Architektura VAX

- Adresní režimy

Addressing mode name	Syntax	Example	Meaning	Length of address specifier in bytes
Literal	#value	#–1	–1	1 (6-bit signed value)
Immediate	#value	#100	100	1 + length of the immediate
Register	rn	r3	r3	1
Register deferred	(rn)	(r3)	Memory[r3]	1
Byte/word/long displacement	Displacement (rn)	100(r3)	Memory[r3 + 100]	1 + length of the displacement
Byte/word/long displacement deferred	@displacement (rn)	@100(r3)	Memory[Memory [r3 + 100]]	1 + length of the displacement
Indexed (scaled)	Base mode [rx]	(r3)[r4]	Memory[r3 + r4 × *d*] (where *d* is data size in bytes)	1 + length of base addressing mode
Autoincrement	(rn)+	(r3)+	Memory[r3]; r3 = r3 + *d*	1
Autodecrement	– (rn)	–(r3)	r3 = r3 – *d;* Memory[r3]	1
Autoincrement deferred	@(rn)+	@(r3)+	Memory[Memory[r3]]; r3 = r3 + *d*	1

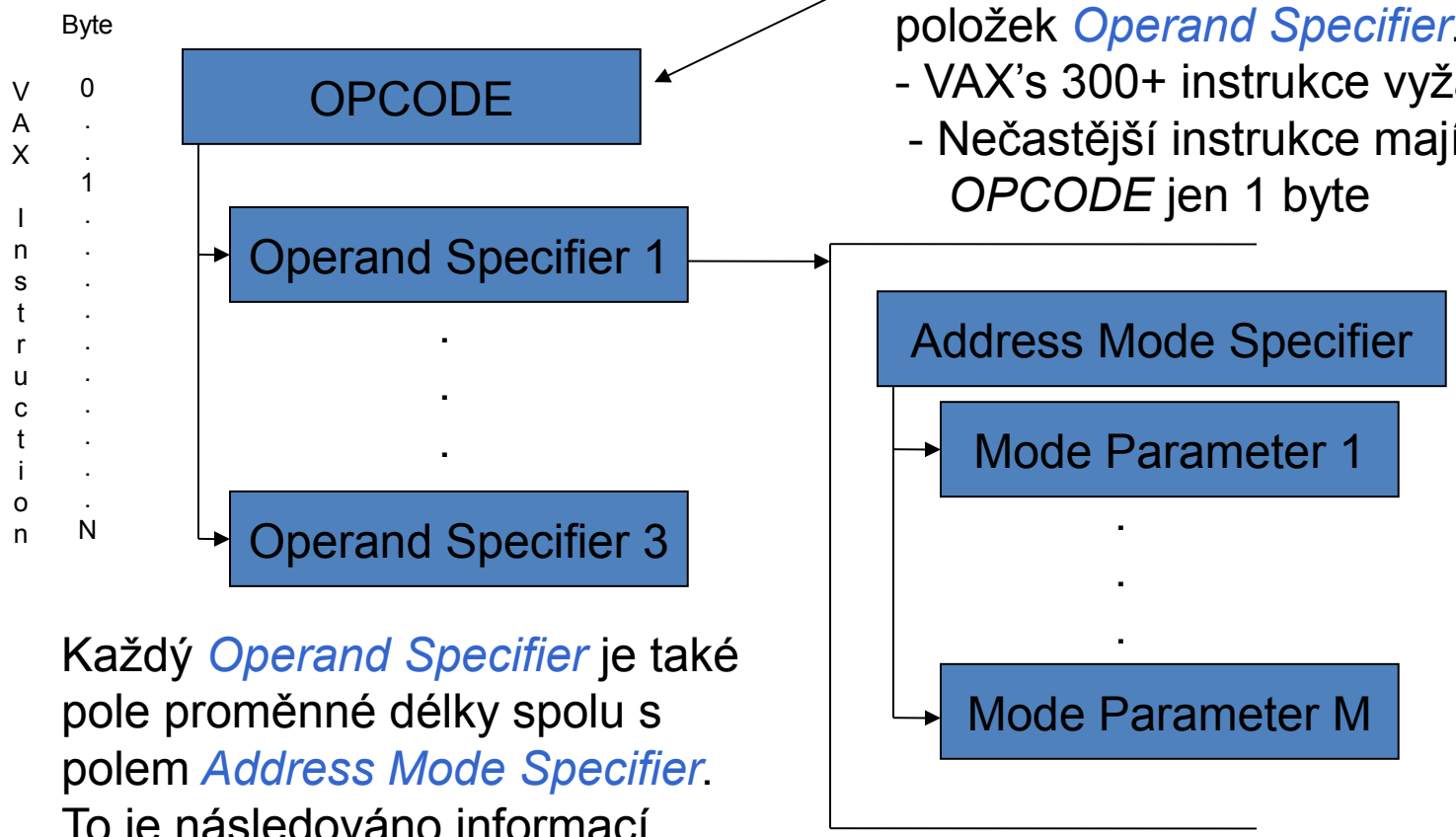
Type	Addressing Mode	Format	Hex Value	Description	Can Be Indexed?
General register	Register	Rn	5	Register contains the operand.	No
	Register deferred	(Rn)	6	Register contains the address of the operand.	Yes
	Autoincrement	(Rn)+	8	Register contains the address of the operand; the processor increments the register contents by the size of the operand data type.	Yes
	Autoincrement deferred	@(Rn)+	9	Register contains the address of the operand address; the processor increments the register contents by 4.	Yes
	Autodecrement	-(Rn)	7	The processor decrements the register contents by the size of the operand data type; the register then contains the address of the operand.	Yes
	Displacement	dis(Rn) B^dis(Rn) W^dis(Rn) L^dis(Rn)	A C E	The sum of the contents of the register and the displacement is the address of the operand; B^, W^, and L^ respectively indicate byte, word, and longword displacement.	Yes
	Displacement deferred	@dis(Rn) @B^dis(Rn) @W^dis(Rn) @L^dis(Rn)	B D F	The sum of the contents of the register and the displacement is the address of the operand address; B^, W^, and L^ respectively indicate, byte, word, and longword displacement.	Yes
	Literal	#literal S^#literal	0-3	The literal specified is the operand; the literal is stored as a short literal.	No
Program counter	Relative	address B^address W^address L^address	A C E	The address specified is the address of the operand; the address is stored as a displacement from the PC; B^, W^, and L^ respectively indicate byte, word, and longword displacement.	Yes

Type	Addressing Mode	Format	Hex Value	Description	Can Be Indexed?
	Relative deferred	@address @B^address @W^address @L^address	B D F	The address specified is the address of the operand address; the address specified is stored as a displacement from the PC; B^, W^, and L^ indicate byte, word, and longword displacement respectively.	Yes
	Absolute	@#address	9	The address specified is the address of the operand; the address specified is stored as an absolute virtual address, not as a displacement.	Yes
	Immediate	#literal I^#literal	8	The literal specified is the operand; the literal is stored as a byte, word, longword, or quadword.	No
	General	G^address	---	The address specified is the address of the operand; if the address is defined as relocatable, the linker stores the address as a displacement from the PC; if the address is defined as an absolute virtual address, the linker stores the address as an absolute value.	Yes
Index	Index	base-mode[Rx]	4	The base-mode specifies the base address and the register specifies the index; the sum of the base address and the product of the contents of Rx and the size of the operand data type is the address of the operand; base mode can be any addressing mode except register, immediate, literal, index, or branch.	No
Branch	Branch	address	---	The address specified is the operand; this address is stored as a displacement from the PC; branch mode can only be used with the branch instructions.	No

Dekódování instrukcí

- Jedno až dvoubytový OPCODE specifikuje operaci, počet operandů a datové typy.
- Poté, co OPCODE určil počet operandů, je každý operand reprezentován položkou Operand Specifier.
- Operand Specifier určuje adresní režim operandu a prvního parametru. Všechny další parametry musí být načítány po svém vyhraženém Operand Specifieru.

Dekódování instrukcí



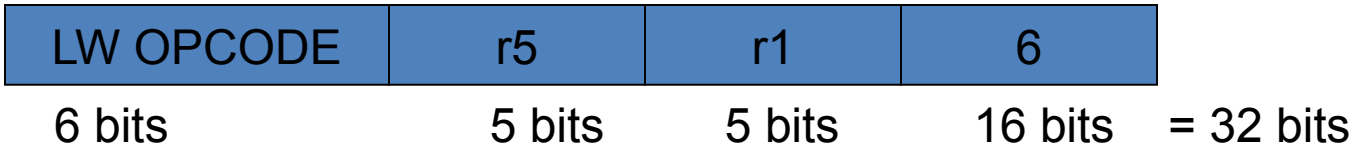
1 až 2 bytový *OPCODE* určuje počet položek *Operand Specifier*.
- VAX's 300+ instrukce vyžadují 9 bitů
- Nečastější instrukce mají délku *OPCODE* jen 1 byte

Každý *Operand Specifier* je také pole proměnné délky spolu s polem *Address Mode Specifier*. To je následováno informací, kterou daný režim vyžaduje.

VAX má proměnnou délku instrukcí

Dekódování instrukcí - LOAD

MIPS: lw r5, 6(r1)



VAX: movl 6(r1), r5

Instrukce Call/Ret

- Instrukce “call” VAXu
  - Multicyklová instrukce
  - Automatizuje a reguluje metody konzervování stavu před voláním procedury
  - Používá uživatelem definovanou masku pro určení registrů, které se mají ukládat
  - Aktualizuje AP a FP aby ukazovaly na parametry aktuálního rámce
  - Aktualizuje PC a spouští provádění procedury
- Instrukce “ret” VAXu
  - Multicyklová instrukce
  - Automatizuje a reguluje obnovu stavu po návratu z procedury
  - Tvoří protějšek k instrukci “call” instrukčního souboru VAXu

Tyto instrukce mohou být značně neefektivní.

Adresní režimy

Immediate

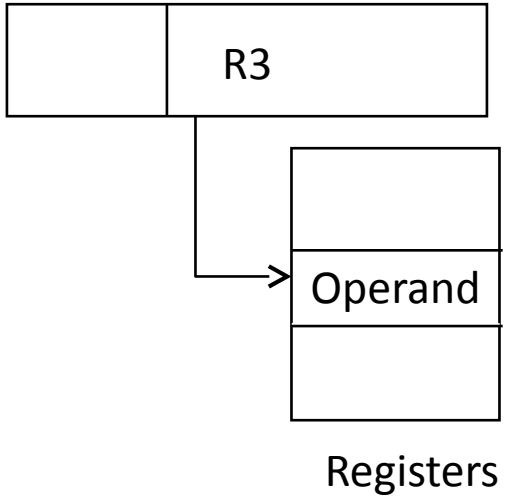
Add R4, #3

Regs[R4] ← Regs[R4]+3

Register

Add R4, R3

Regs[R4] ← Regs[R4]+Regs[R3]

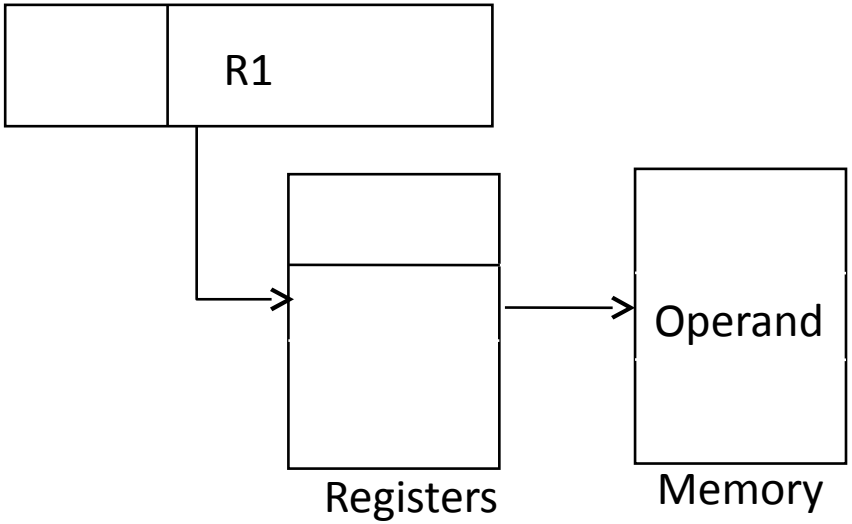


Adresní režimy

Register Indirect

Add R4, (R1)

Regs[R4] ← Regs[R4]+Mem[Regs[R1]]

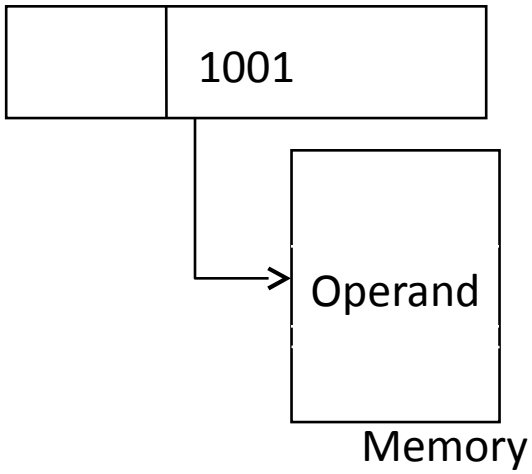


Adresní režimy

Direct

Add R4, (1001)

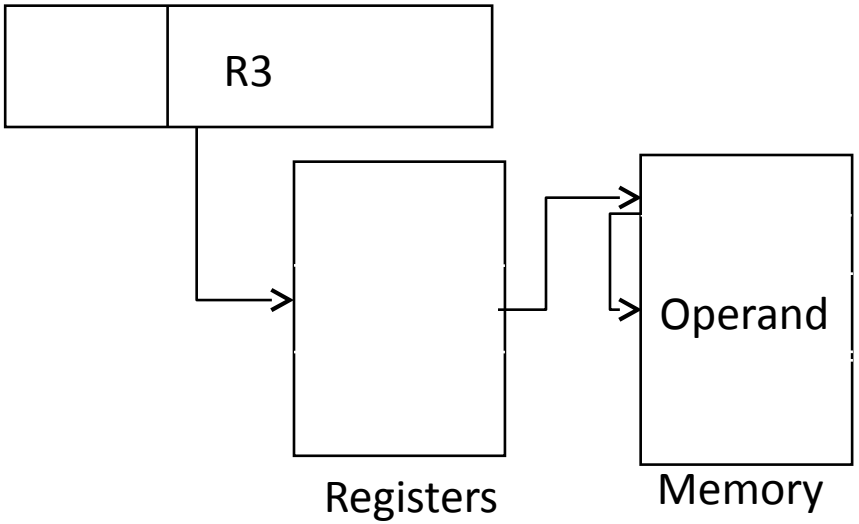
Regs[R4] ← Regs[R4]+Mem[1001]



Memory Indirect

Add R4, @(R3)

Regs[R4] ← Regs[R4]+Mem[Mem[Regs[R3]]]

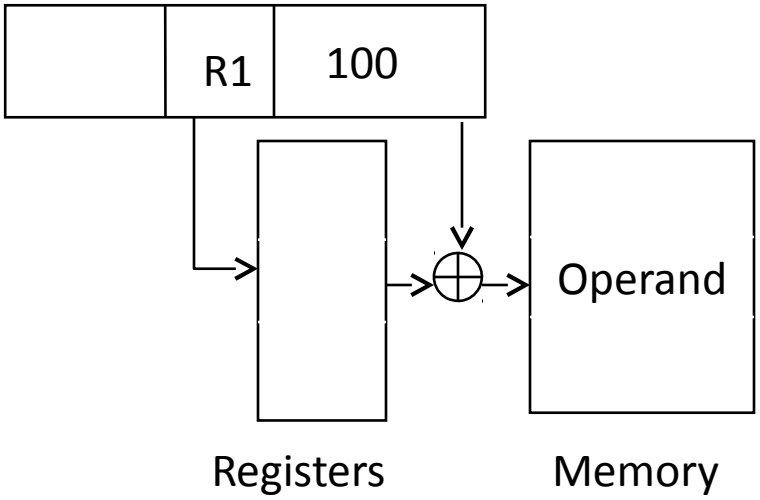


Adresní režimy

Displacement

Add R4, 100(R1)

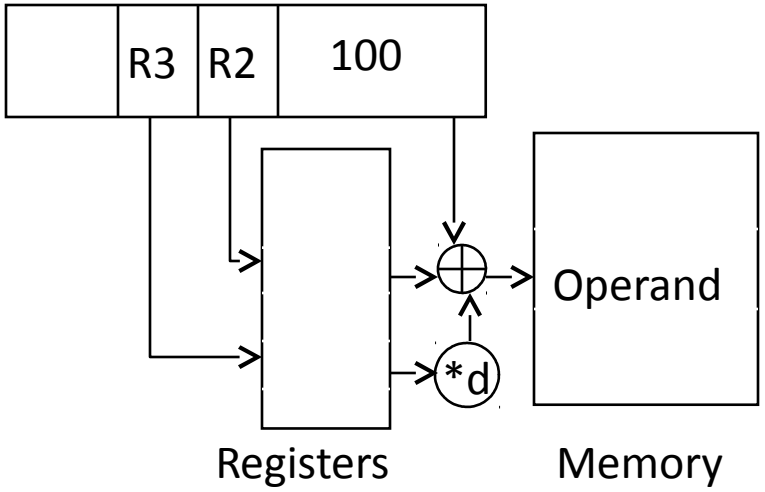
Regs[R4] ← Regs[R4]+Mem[100+R1]



Scaled

Add R1, 100(R2) [R3]

Regs[R1] ← Regs[R1]+Mem[100+
Regs[R2]+Regs[R3]*d]



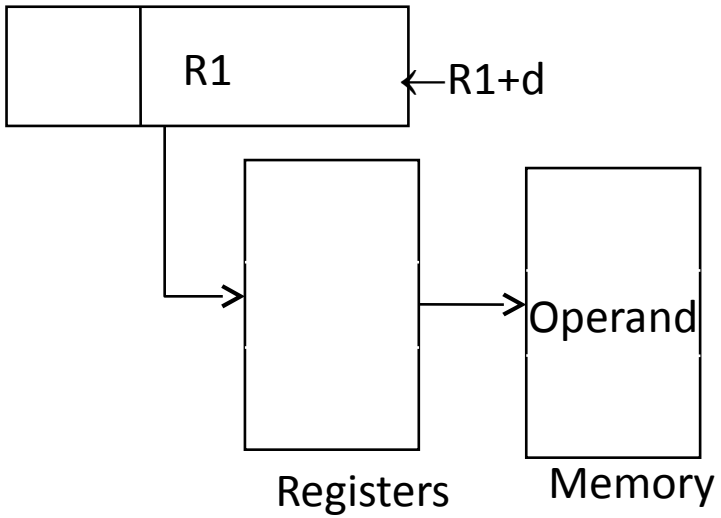
Adresní režimy

Autoincrement

Add R4, (R1)+

Regs[R4] ← Regs[R4]+Mem[Regs[R1]]

Regs[R1] ← Regs[R1]+d

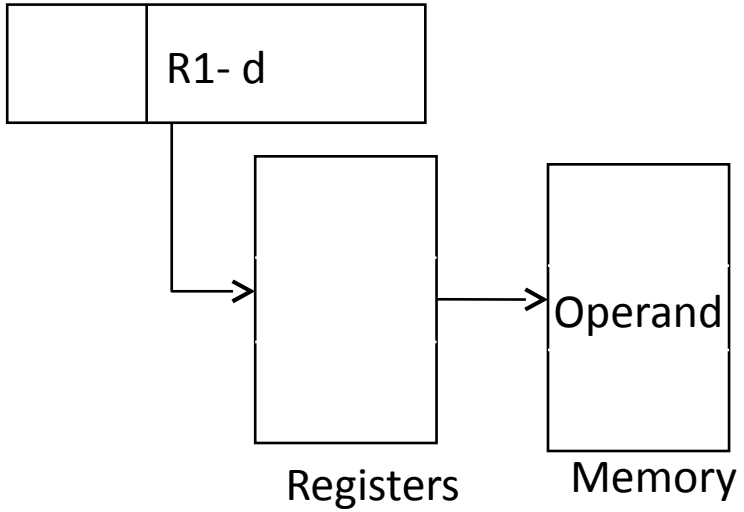


Autodecrement

Add R4, (R1)-

Regs[R1] ← Regs[R1]-d

Regs[R4] ← Regs[R4]+Mem[Regs[R1]]

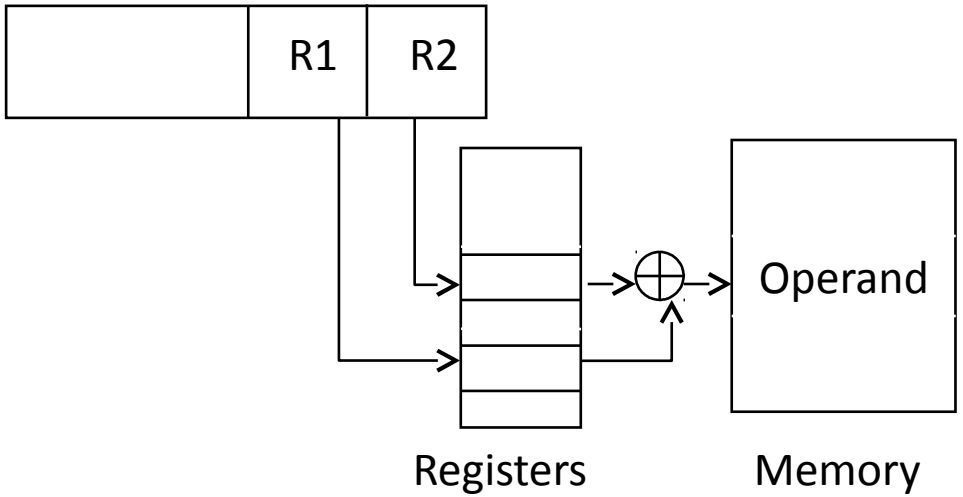


Adresní režimy

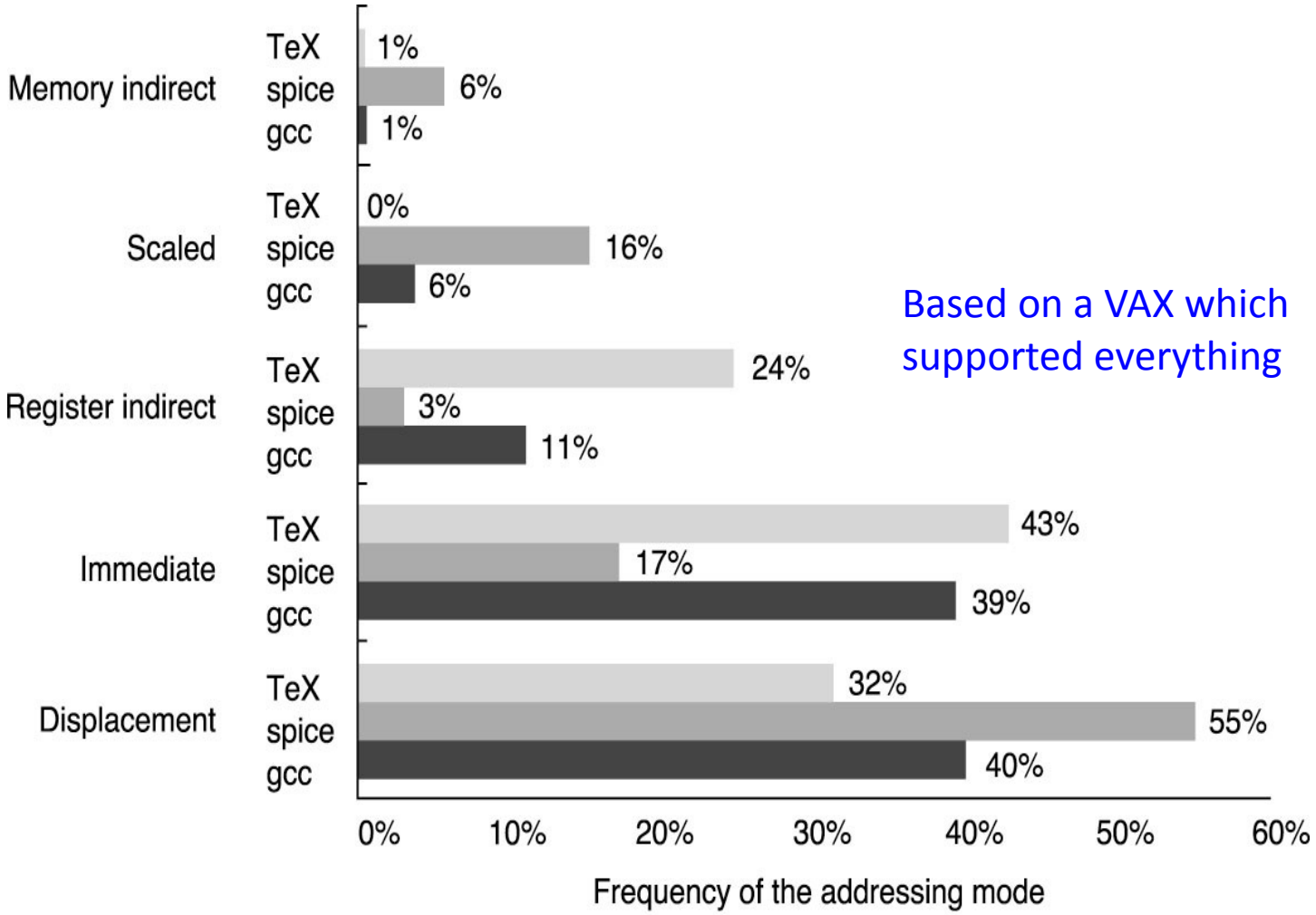
Indexed

Add R4, (R1+R2)

Regs[R4] ← Regs[R4]+Mem[Regs[R1]+Regs[R2]]



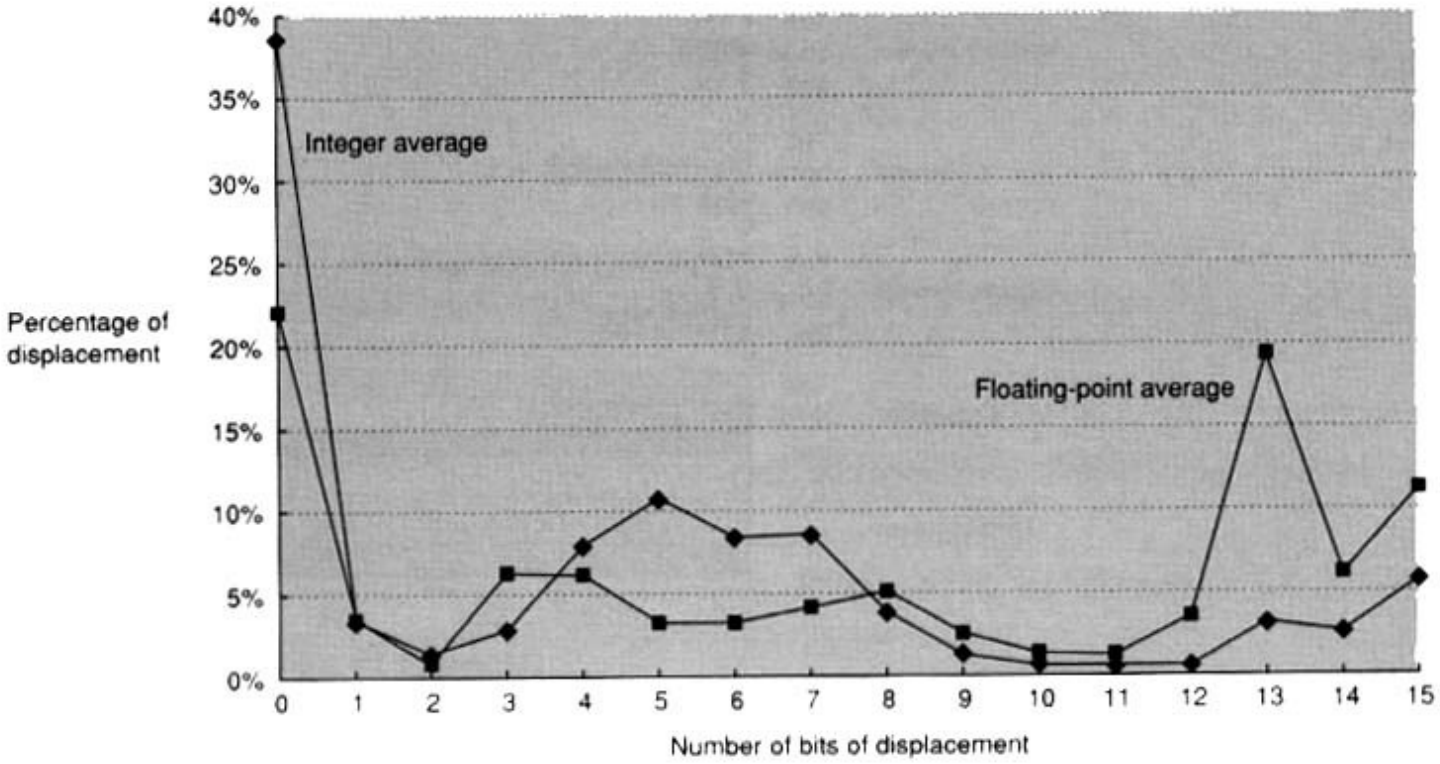
Adresní režimy - využití



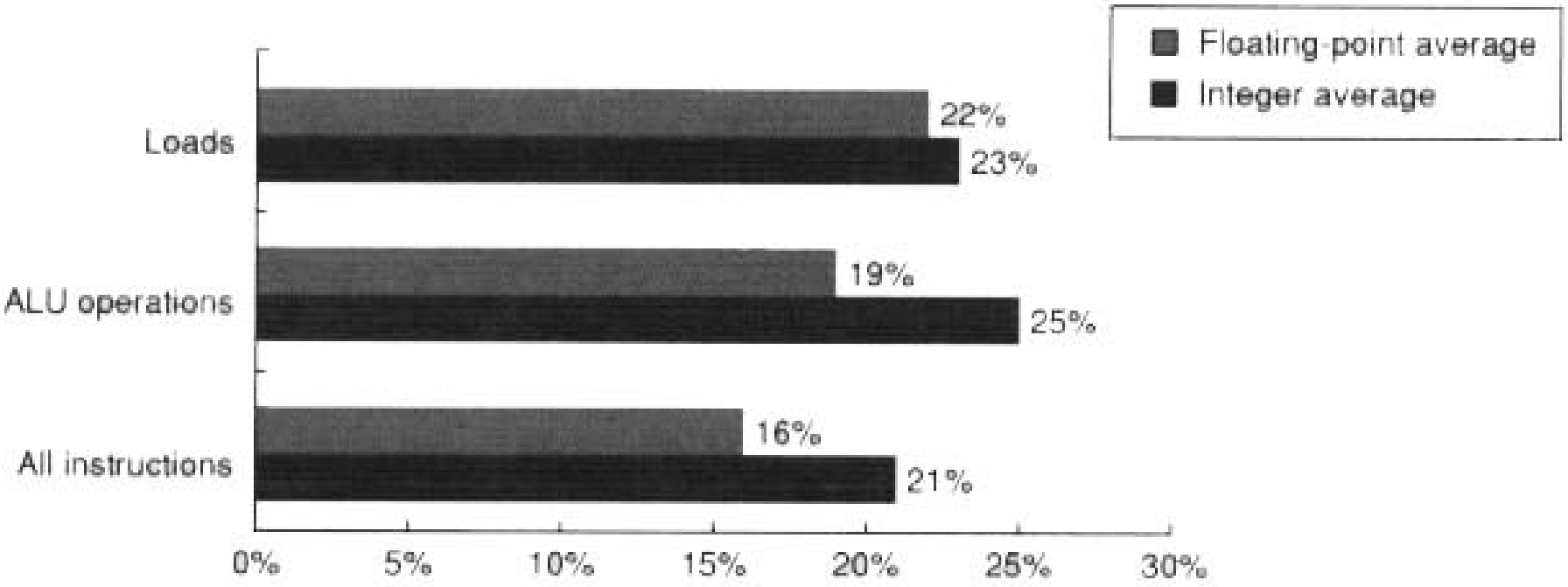
Based on a VAX which supported everything

Velikost displacementu

- Průměr z SPEC CINT2000 a SPEC CFP2000.
  - Osa X je $\log_2$ displacementu.
  - 1% adres > 16 bitů.

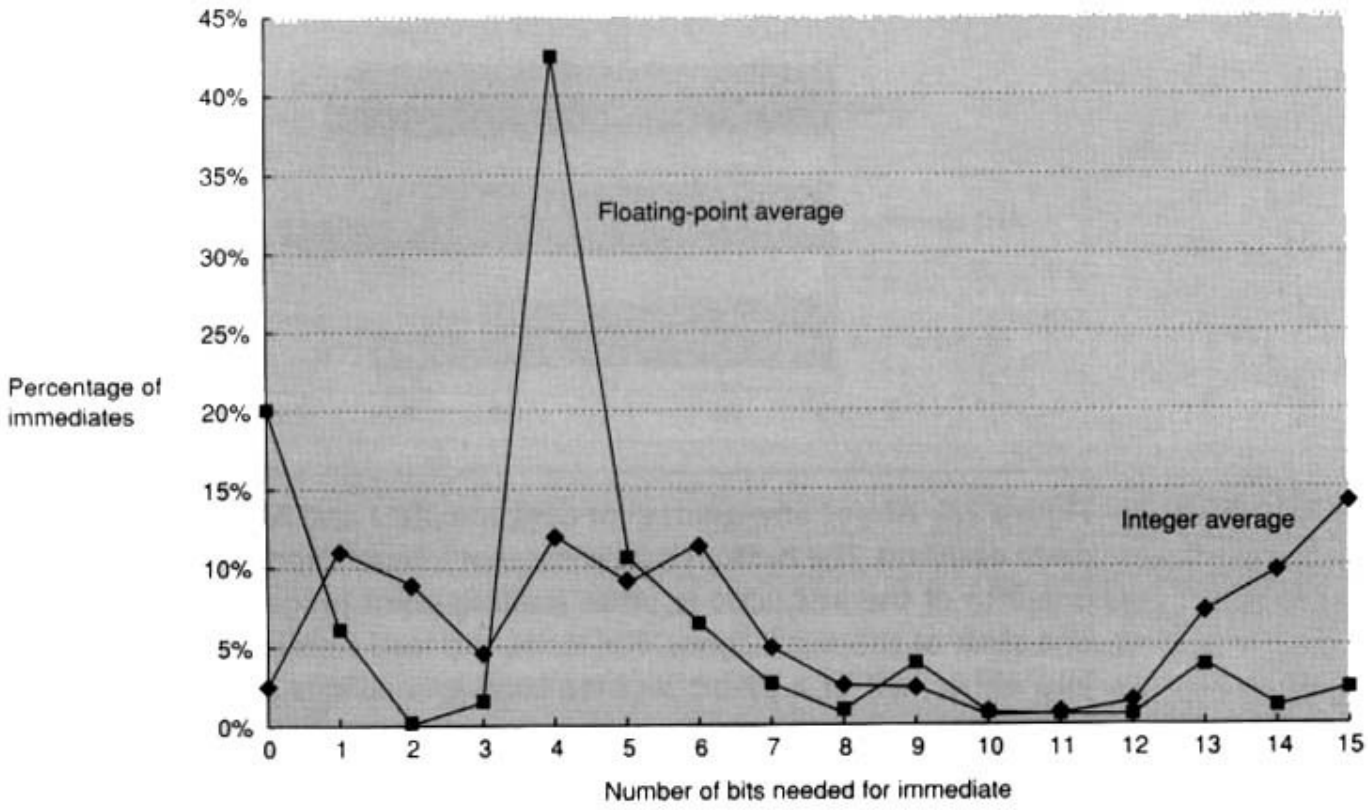


Adresní režim - immediate



Adresní režim - immediate

- 50% až 60% vystačí s délkou 8 bitů
- 75% až 80% vystačí s délkou 16 bitů

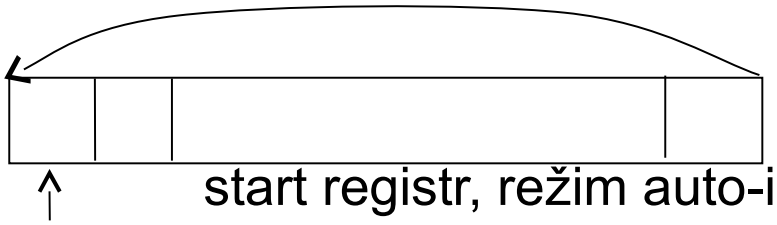


Adresní režimy - shrnutí

- Důležité adresní režimy
  - Displacement
  - Immediate
  - Register Indirect
- Položka displacement by měla být 12 až 16 bitů.
- Velikost přímého operandu (immediate) by měla být 8 až 16 bitů.

Adresní režimy pro zpracování signálů

- Použití DSP
  - Zpracování kontinuálního proudu dat několika základními operacemi – založeno na využití kruhového bufferu. Vedle několika základních adresních módů (např. immediate, displacement, register indirect, direct, atd.) se používají i dva následující adresní módy
  - Modulo (kruhový) adresní režim



kruhový buffer

start registr, režim auto-inkrement (nebo dekrement)

Reverzní adresní režim

(Shuffled data access pro FFT)

0 (000) ⇒ 0 (000)

1 (001) ⇒ 4 (100)

2 (010) ⇒ 2 (010)

3 (011) ⇒ 6 (110)

4 (100) ⇒ 1 (001)

5 (101) ⇒ 5 (101)

6 (110) ⇒ 3 (011)

7 (111) ⇒ 7 (111)

Od adresy 1 do 4, invertujeme každý bit adresy

- Příklad četnosti výskytu adresních módů u TI TMS320C54x DSP

Četnost výskytu adresních módů pro TI TMS320C54x DSP.

Addressing mode	Assembly symbol	Percent
Immediate	#num	30.02%
Displacement	ARx(num)	10.82%
Register indirect	*ARx	17.42%
Direct	num	11.99%
Autoincrement, preincrement (increment register *before* using contents as address)	*+ARx	0
Autoincrement, postincrement (increment register *after* using contents as address)	*ARx+	18.84%
Autoincrement, preincrement with 16b immediate	*+ARx(num)	0.77%
Autoincrement, preincrement, with circular addressing	*ARx+%	0.08%
Autoincrement, postincrement with 16b immediate, with circular addressing	*ARx+(num)%	0
Autoincrement, postincrement by contents of AR0	*ARx+0	1.54%
Autoincrement, postincrement by contents of AR0, with circular addressing	*ARx+0%	2.15%
Autoincrement, postincrement by contents of AR0, with bit reverse addressing	*ARx+0B	0
Autodecrement, postdecrement (decrement register *after* using contents as address)	*ARx-	6.08%
Autodecrement, postdecrement, with circular addressing	*ARx-%	0.04%
Autodecrement, postdecrement by contents of AR0	*ARx-0	0.16%
Autodecrement, postdecrement by contents of AR0, with circular addressing	*ARx-0%	0.08%
Autodecrement, postdecrement by contents of AR0, with bit reverse addressing	*ARx-0B	0
Total		100.00%

Třídy instrukcí

Operator type	Examples
Arithmetic and logical	Integer arithmetic and logical operations: add, subtract, and, or, multiple, divide
Data transfer	Loads-stores (move instructions on computers with memory addressing)
Control	Branch, jump, procedure call and return, traps
System	Operating system call, virtual memory management instructions
Floating point	Floating-point operations: add, multiply, divide, compare
Decimal	Decimal add, decimal multiply, decimal-to-character conversions
String	String move, string compare, string search
Graphics	Pixel and vertex operations, compression/decompression operations

Top-9 instrukcí x86

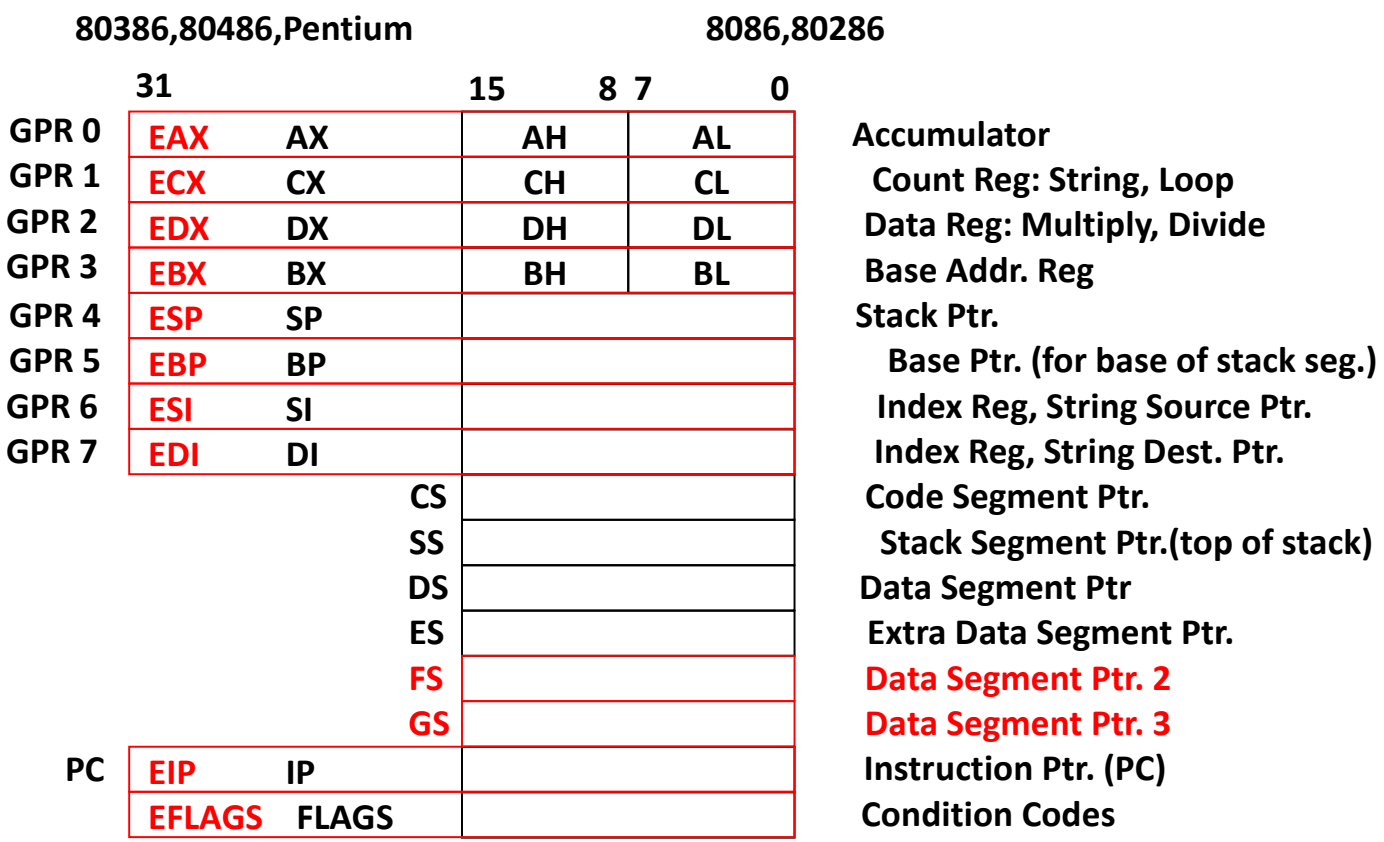
1	Load	
2	Conditic	
3	Compar	
4	Store	

- Jednoduché instrukce dominují co do četnosti.

Adresní režimy MIPS

- Adresování dat: immediate a displacement (16 bitů)
  - **Displacement**: Add R4, 100(R1)
    (Regs[R4]←Regs[R4]+Mem[100+Regs[R1]])
  - **Register-indirect**: umístění 0 do pole displacement
    - Add R4, (R1) (Regs[R4]←Regs[R4]+Mem[Regs[R1]])
  - **Absolutní adresování** (16 bitů): R0 je použit jako bázový register
    - Add R4, (1001) (Regs[R4]←Regs[R4]+Mem[1001])
- Adresování byte 64-bitovou adresou
  - Výběr módu pro Big Endian nebo Little Endian

Registry integer Intel 80x86



Sedm adresních módů pro data

Adresní režimy: kde jsou operandy?

- **Immediate (register + constant)**

```
MOV   EAX,10      ; EAX = 10
```

- **Direct (offset address + register)**

```
MOV   EAX,DATA5   ; EAX = Mem[&DATA5]
MOV   HERE, AL
```

- **Register (register + register)**

```
MOV   EAX,EBX ; EAX = EBX
```

- **Register indirect ([register] + register)**

```
MOV   EAX,[EBX]   ; EAX = Memory[EBX]
```

- **Based with 8- or 32-bit displacement**

  ([register ± offset] or offset[register] + register)

```
MOV   EAX,[EBX+8] ; EAX = Mem[EBX+8]
```

- **Based with scaled index (scale = 0 .. 3)**

  (register + [sf*register] (sf : scale factor))

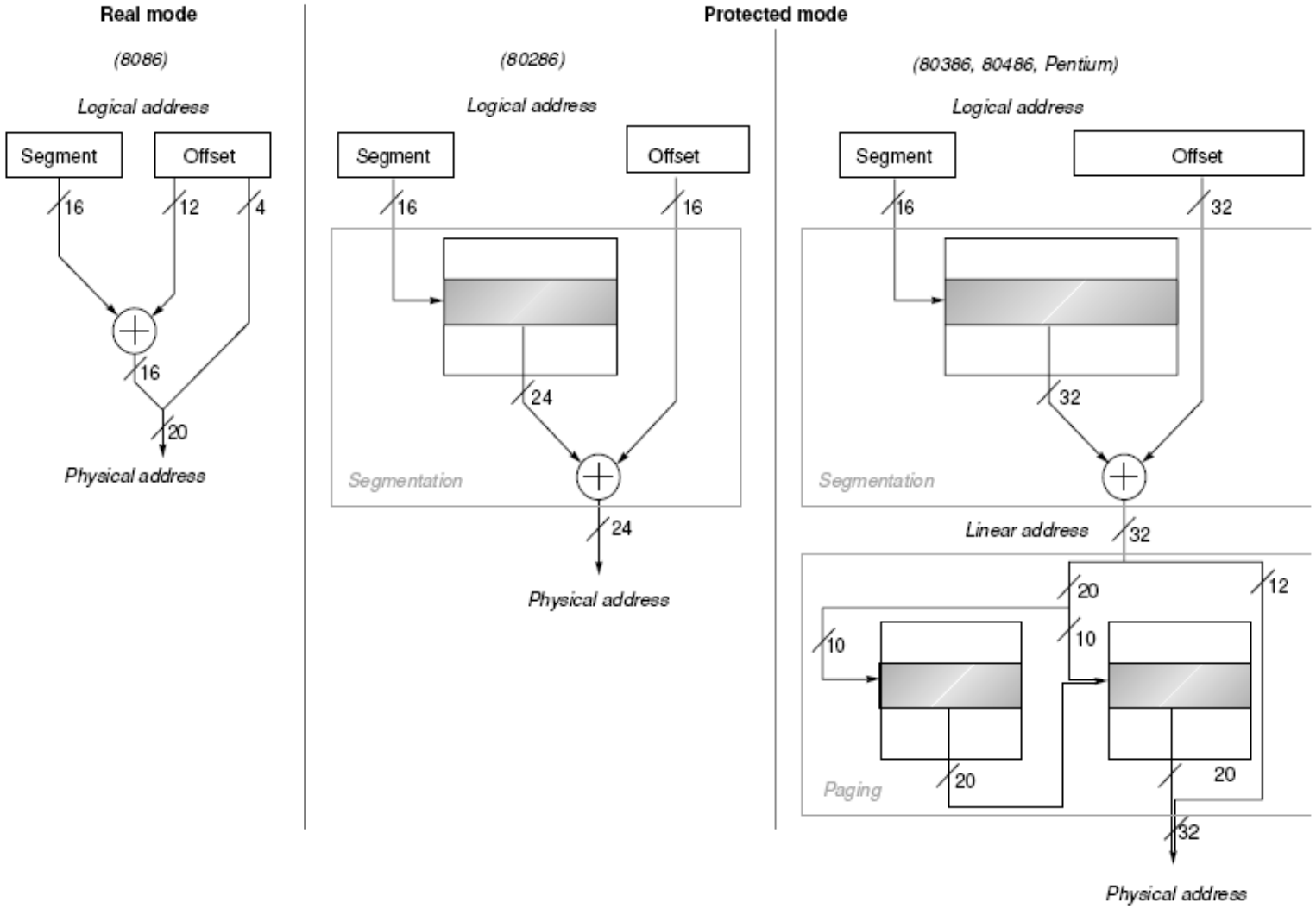
```
MOV   EAX,ECX[EBX] ; EAX = Mem[EBX + 2^scale * ECX]
```

- **Based plus scaled index with 8- or 32-bit displacement**

  (register + [sf*register + offset] or offset[sf*register])

```
MOV   EAX,ECX[EBX+8]
```

Adresování a ochrana 80x86



Typické operace 80x86

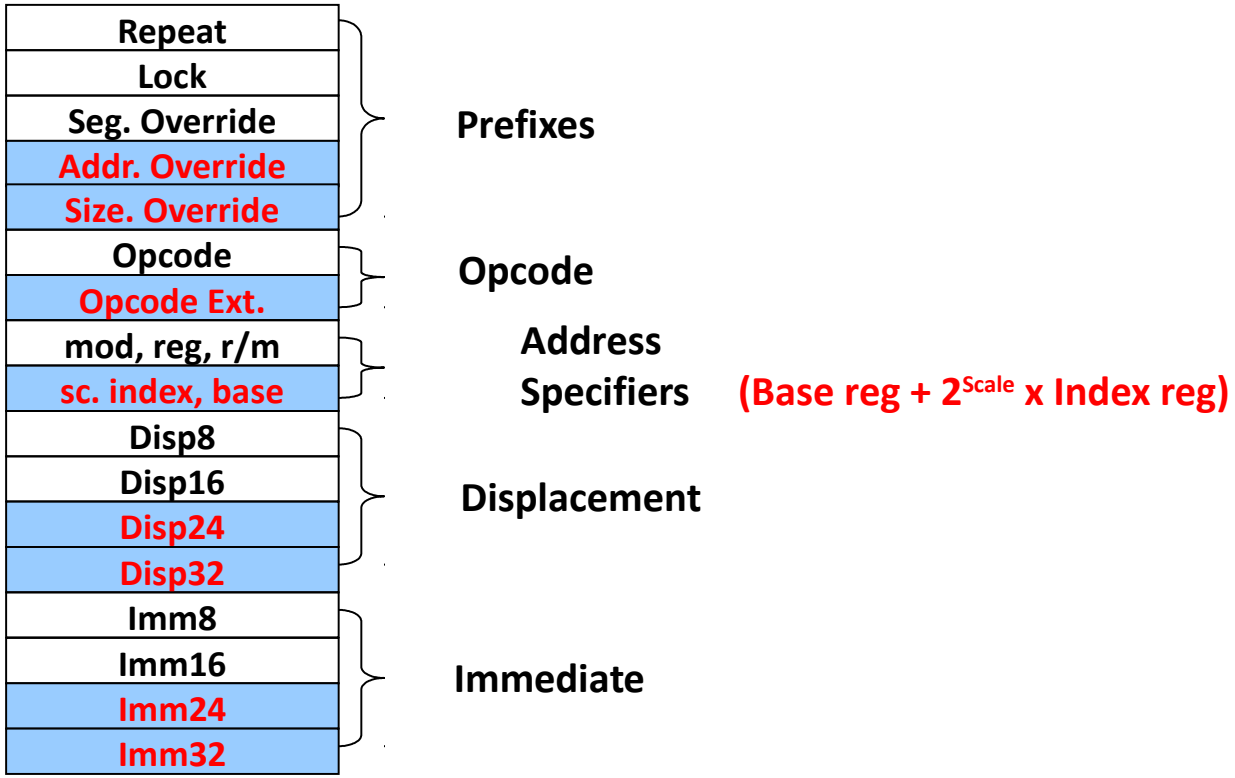
Instruction	Meaning
Control	**Conditional and unconditional branches**
JNZ, JZ	Jump if condition to IP + 8-bit offset; JNE (for JNZ), JE (for JZ) are alternative names
JMP, JMPF	Unconditional jump – 8- or 16-bit offset intrasegment (near), and intersegment (far) versions
CALL, CALLF	Subroutine call – 16-bit offset; return address pushed; near and far versions
RET, RETF	Pops return address from stack and jumps to it; near and far versions
LOOP	Loop branch – decrement CX; jump to IP + 8-bit displacement if CX≠0
Data transfer	**Move data between registers or between register and memory**
MOV	Move between two registers or between register and memory
PUSH	Push source operand on stack
POP	Pop operand from stack top to a register
LES	Load ES and one of the GPRs from memory
String instructions	**Move between string operands; length given by a repeat prefix**
MOVS	Copies from string source to destination; may be repeated
LODS	Loads a byte or word of a string into the A register

Typické operace 80x86

Instruction	Meaning
Arithmetic/logical	Arithmetic and logical operations using the data registers and memory
ADD	Add source to destination; register–memory format
SUB	Subtract source from destination; register–memory format
CMP	Compare source and destination; register–memory format
SHL	Shift left
SHR	Shift logical right
RCR	Rotate right with carry as fill
CBW	Convert byte in AL to word in AX
TEST	Logical AND of source and destination sets flags
INC	Increment destination; register–memory format
DEC	Decrement destination; register–memory format
OR	Logical OR; register–memory format
XOR	Exclusive OR; register–memory format

Instrukční formát 80x86

- 8086 černě : rozšíření 80386 barevně



Dekódování instrukcí 80x86 : Pole Mod, Reg, R/M

		w = 1			mod = 0		mod = 1		mod = 2		
reg	w = 0	16b	32b	r/m	16b	32b	16b	32b	16b	32b	mod = 3
0	AL	AX	EAX	0	addr=BX+SI	=EAX	*same*	*same*	*same*	*same*	*same*
1	CL	CX	ECX	1	addr=BX+DI	=ECX	*addr as*	*addr as*	*addr as*	*addr as*	*as*
2	DL	DX	EDX	2	addr=BP+SI	=EDX	*mod=0*	*mod=0*	*mod=0*	*mod=0*	*reg*
3	BL	BX	EBX	3	addr=BP+SI	=EBX	*+ disp8*	*+ disp8*	*+ disp16*	*+ disp32*	*field*
4	AH	SP	ESP	4	addr=SI	=(sib)	SI+disp16	(sib)+disp8	SI+disp8	(sib)+disp32	"
5	CH	BP	EBP	5	addr=DI	=disp32	DI+disp8	EBP+disp8	DI+disp16	EBP+disp32	"
6	DH	SI	ESI	6	addr=disp16	=ESI	BP+disp8	ESI+disp8	BP+disp16	ESI+disp32	"
7	BH	DI	EDI	7	addr=BX	=EDI	BX+disp8	EDI+disp8	BX+disp16	EDI+disp32	"

Dekódování instrukcí 80x86 : Pole Sc/Index/Base

	Index	Base
0	EAX	EAX
1	ECX	ECX
2	EDX	EDX
3	EBX	EBX
4	No index	ESP
5	EBP	If mod =0, d32 If mod≠0, EBP
6	ESI	ESI
7	EDI	EDI

Base + Scaled Index Mode

Used when:

mod = 0, 1, 2

in 32-bit mode

AND r/m = 4!

2-bit Scale Field

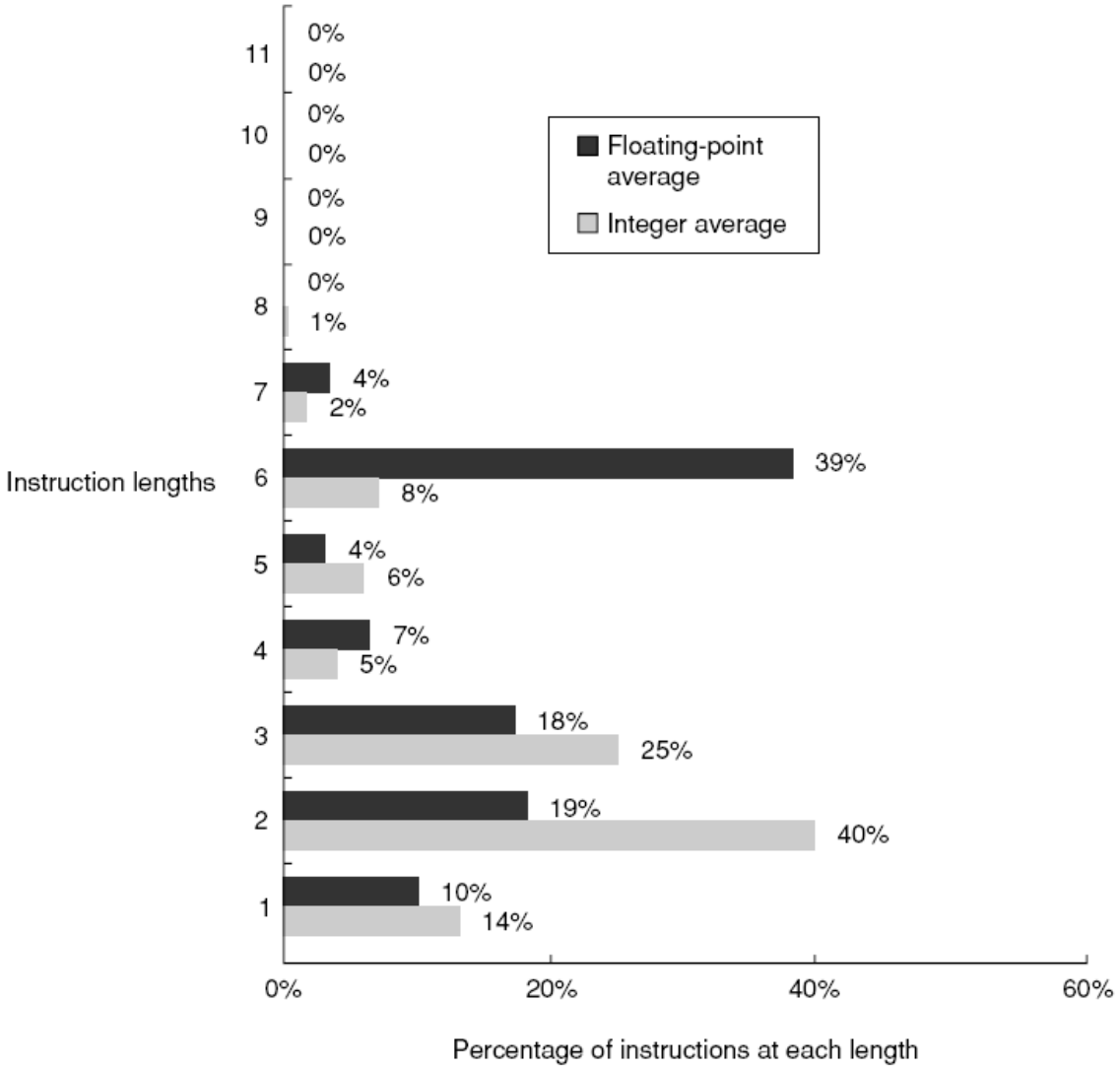
3-bit Index Field

3-bit Base Field

Četnost výskytu adresních režimů v programech

Addressing Mode	Integer average	FP average
Register indirect	13%	3%
Base + 8-bit disp	31%	15%
Base + 32-bit disp	9%	25%
Indexed	0%	0%
Base Indexed+8b disp	0%	0%
Base Indexed+32b disp	0%	1%
Base+Scaled Indexed	22%	7%
Base+Scaled Index+8b disp	0%	8%
Base+Scaled Index+32b disp	4%	4%
32-bit Direct	20%	37%

Histogram délek instrukcí 80x86



Zastoupení FP instrukcí

Option	doduc	ear	hydro2d	mdljdp2	su2cor	FP average
Stack (2nd operand ST (1))	1.1%	0.0%	0.0%	0.2%	0.6%	0.4%
Register (2nd operand ST(i), i > 1)	17.3%	63.4%	14.2%	7.1%	30.7%	26.5%
Memory	81.6%	36.6%	85.8%	92.7%	68.7%	73.1%

Závěr

- Komplexní adresní režimy a velký počet instrukčních formátů vyžadují složité řízení.
- Obtížně řešitelné v systémech s hlubokým pipelinningem.