计算机指令

Part4：ARM指令集、X86指令集

本讲提纲

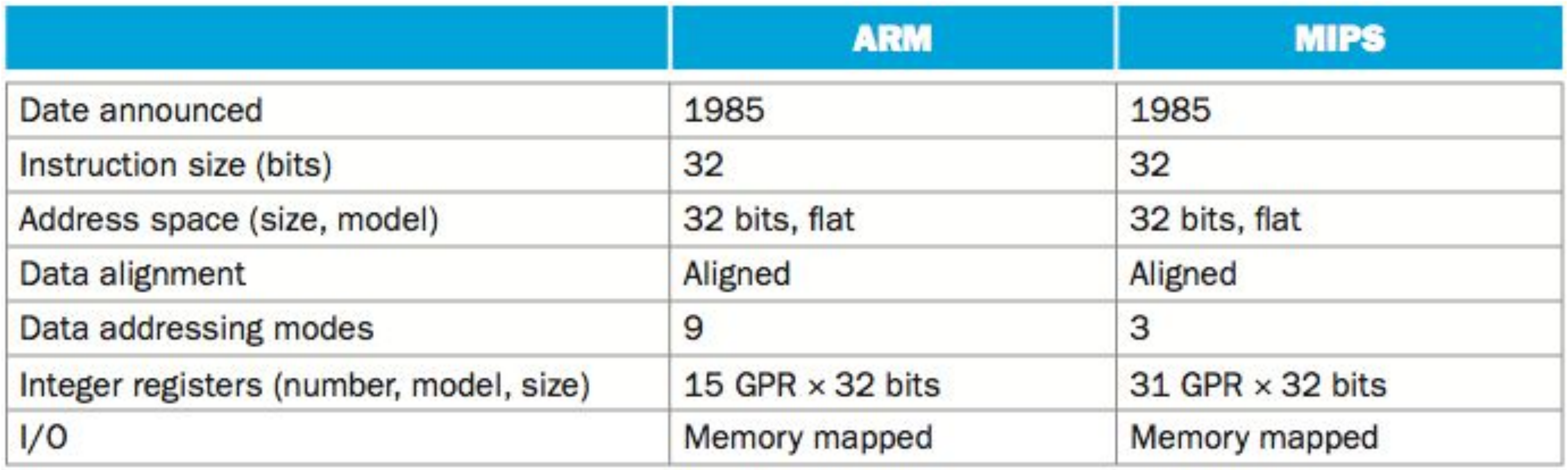
- 指令集实例
  - ARM
  - x86

指令集实例

- ARM指令集
  - 32位精简指令集（RISC）处理器架构，其广泛地使用在许多嵌入式系统设计。由于节能的特点，ARM处理器非常适用于移动通信领域，符合其主要设计目标为低成本、高性能、低耗电的特性
  - ARM处理器：Apple A5、nVIDIA Tegra 4
- X86指令集
  - 复杂指令集处理器架构
  - x86架构于1978年推出的Intel 8086中央处理器中首度出现，它是从Intel 8008处理器中发展而来的，而8008则是发展自Intel 4004的。
  - 8086是16位处理器；直到1985年32位的80386的开发，这个架构都维持是16位。接着一系列的处理器进行了32位架构的细微改进。
  - 直到2003年AMD对于这个架构发展了64位的扩充，并命名为AMD64。后来英特尔也推出了与之兼容的处理器，并命名为Intel 64。两者一般被统称为x86-64或x64，开创了x86的64位时代。

ARM指令集

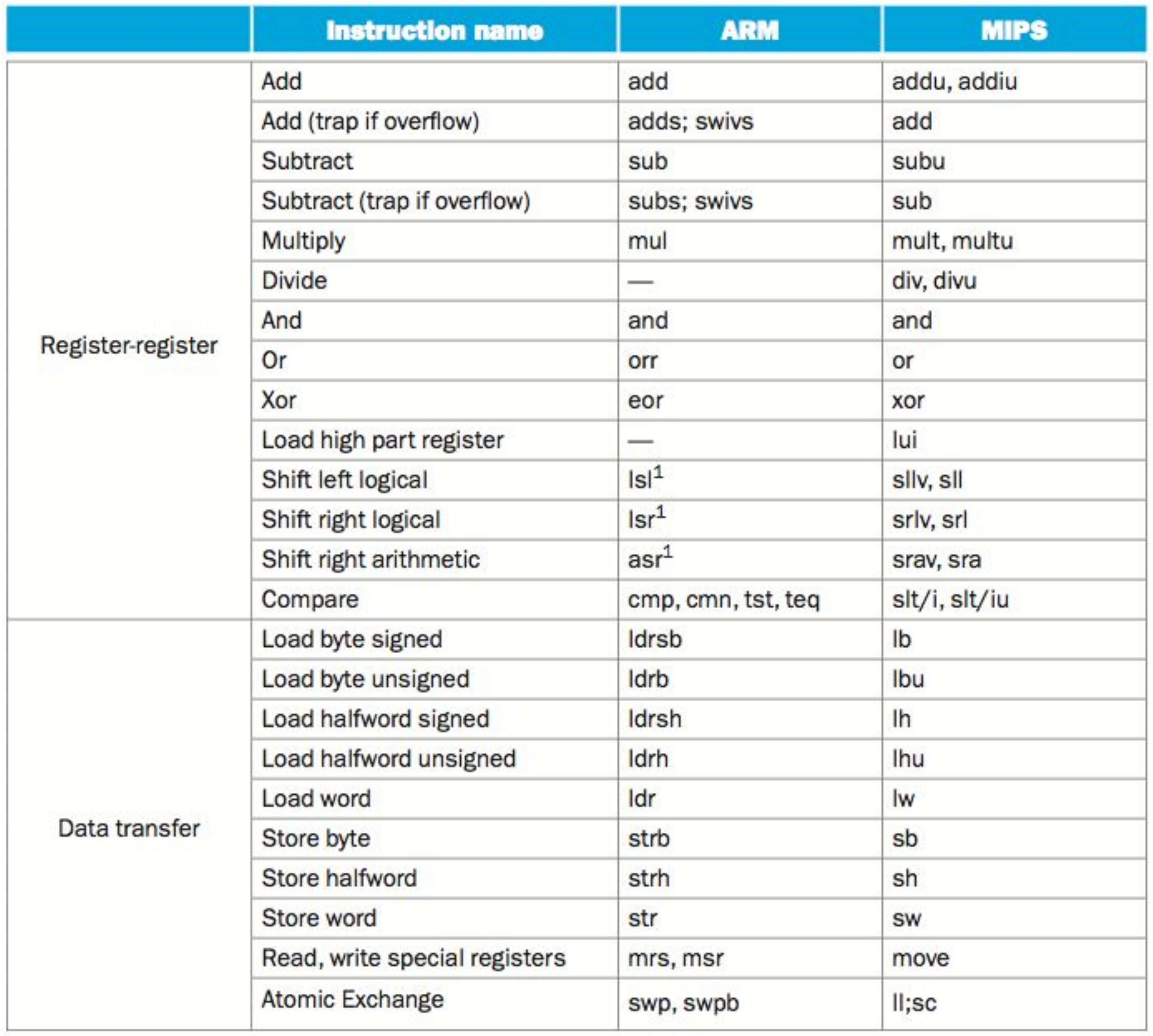
- 嵌入式设备领域最流行的指令集体系结构
- 最初代表Acorn RISC Machine，后改为Advanced RISC Machine
- 与MIPS相比，具有较少的寄存器，更多的寻址模式



	ARM	MIPS
Date announced	1985	1985
Instruction size (bits)	32	32
Address space (size, model)	32 bits, flat	32 bits, flat
Data alignment	Aligned	Aligned
Data addressing modes	9	3
Integer registers (number, model, size)	15 GPR × 32 bits	31 GPR × 32 bits
I/O	Memory mapped	Memory mapped

ARM指令集

- ARM的寄存器-寄存器指令和数据传输指令和MIPS是等价的



	Instruction name	ARM	MIPS
Register-register	Add	add	addu, addiu
	Add (trap if overflow)	adds; swivs	add
	Subtract	sub	subu
	Subtract (trap if overflow)	subs; swivs	sub
	Multiply	mul	mult, multu
	Divide	—	div, divu
	And	and	and
	Or	orr	or
	Xor	eor	xor
	Load high part register	—	lui
	Shift left logical	lsl[1]	sllv, sll
	Shift right logical	lsr[1]	srlv, srl
	Shift right arithmetic	asr[1]	srav, sra
	Compare	cmp, cmn, tst, teq	slt/i, slt/iu
Data transfer	Load byte signed	ldrsb	lb
	Load byte unsigned	ldrb	lbu
	Load halfword signed	ldrsh	lh
	Load halfword unsigned	ldrh	lhu
	Load word	ldr	lw
	Store byte	strb	sb
	Store halfword	strh	sh
	Store word	str	sw
	Read, write special registers	mrs, msr	move
	Atomic Exchange	swp, swpb	ll;sc

ARM指令集

- 寻址模式

Addressing mode	ARM v.4	MIPS
寄存器操作数	X	X
立即数操作数	X	X
寄存器+偏移（转移或基地址）	X	X
寄存器+寄存器（下标）	X	—
寄存器+寄存器倍乘（倍乘）	X	—
寄存器+偏移和更新寄存器	X	—
寄存器+寄存器和更新寄存器	X	—
自增、自减	X	—
相对PC的数据	X	—

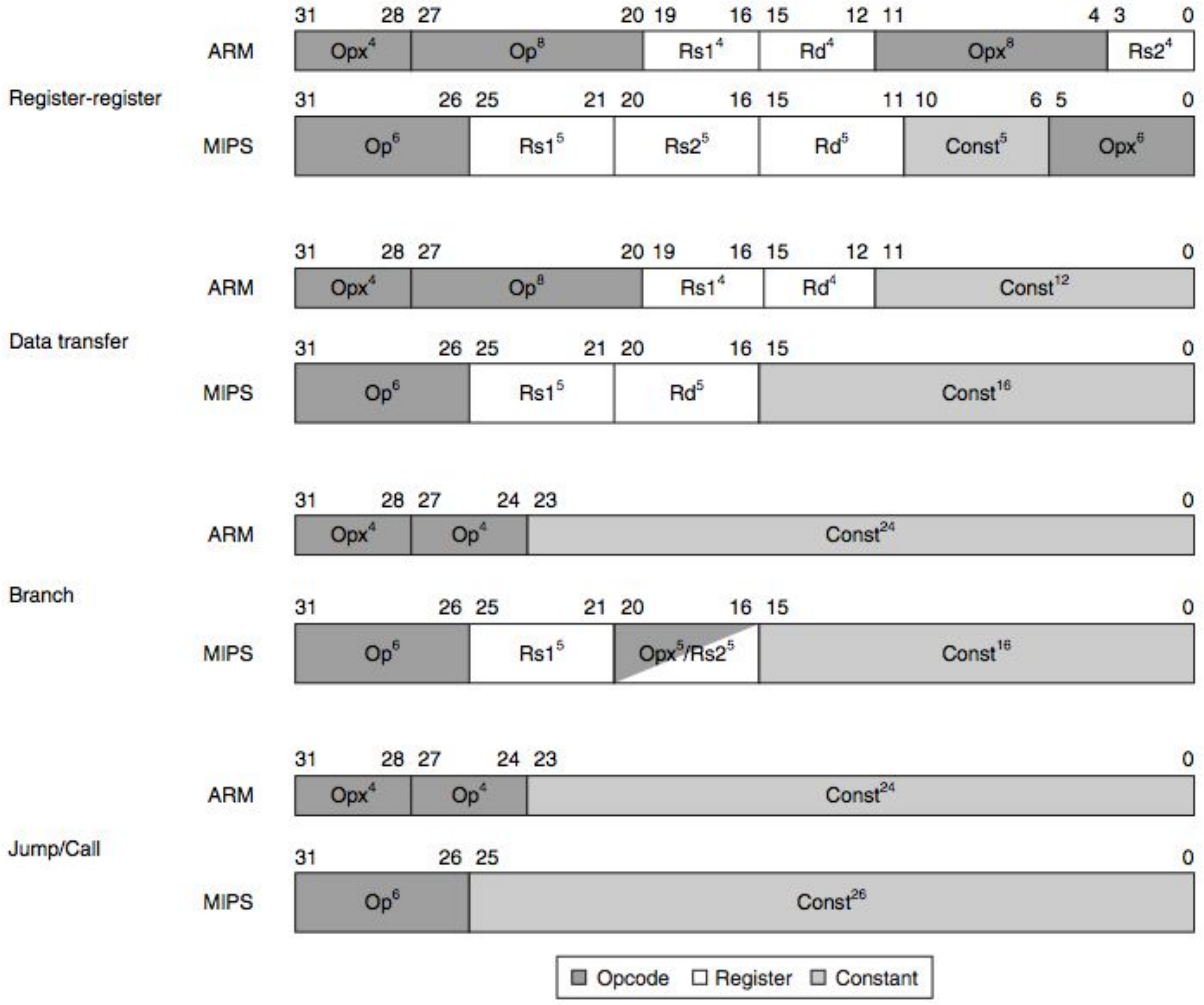
  - ARM具有分离的寄存器间接寻址和寄存器+偏移寻址模式
  - 为了增加寻址范围，如果是对半字或字进行操作，ARM对偏移左移1位或2位

ARM指令格式

- 主要区别

每条指令的4位条件执行字段不同；

ARM拥有较小的寄存器字段。



ARM的特点

Name	Definition	ARM v.4	MIPS
取立即数	Rd = Imm	mov	addi, $0,
非	Rd = ~(Rs1)	mvn	nor, $0,
移动	Rd = Rs1	mov	or, $0,
右旋转	Rd = Rs i >> i $Rd_{0...i-1} = Rs_{31-i...31}$	ror	
和寄存器非的与	Rd = Rs1 & ~(Rs2)	bic	
反向减	Rd = Rs2 - Rs1	rsb, rsc	
支持多个整数字的加	CarryOut, Rd = Rd + Rs1 + OldCarryOut	adcs	—
支持多个整数字的减	CarryOut, Rd = Rd – Rs1 + OldCarryOut	sbcs	—

MIPS中没有的ARM算术/逻辑指令

ARM的特点

- 没有专门的寄存器存储0
- 对操作数的移位并不限于立即数
- ARM还对寄存器组的操作提供了指令支持
    - 块加载和存储

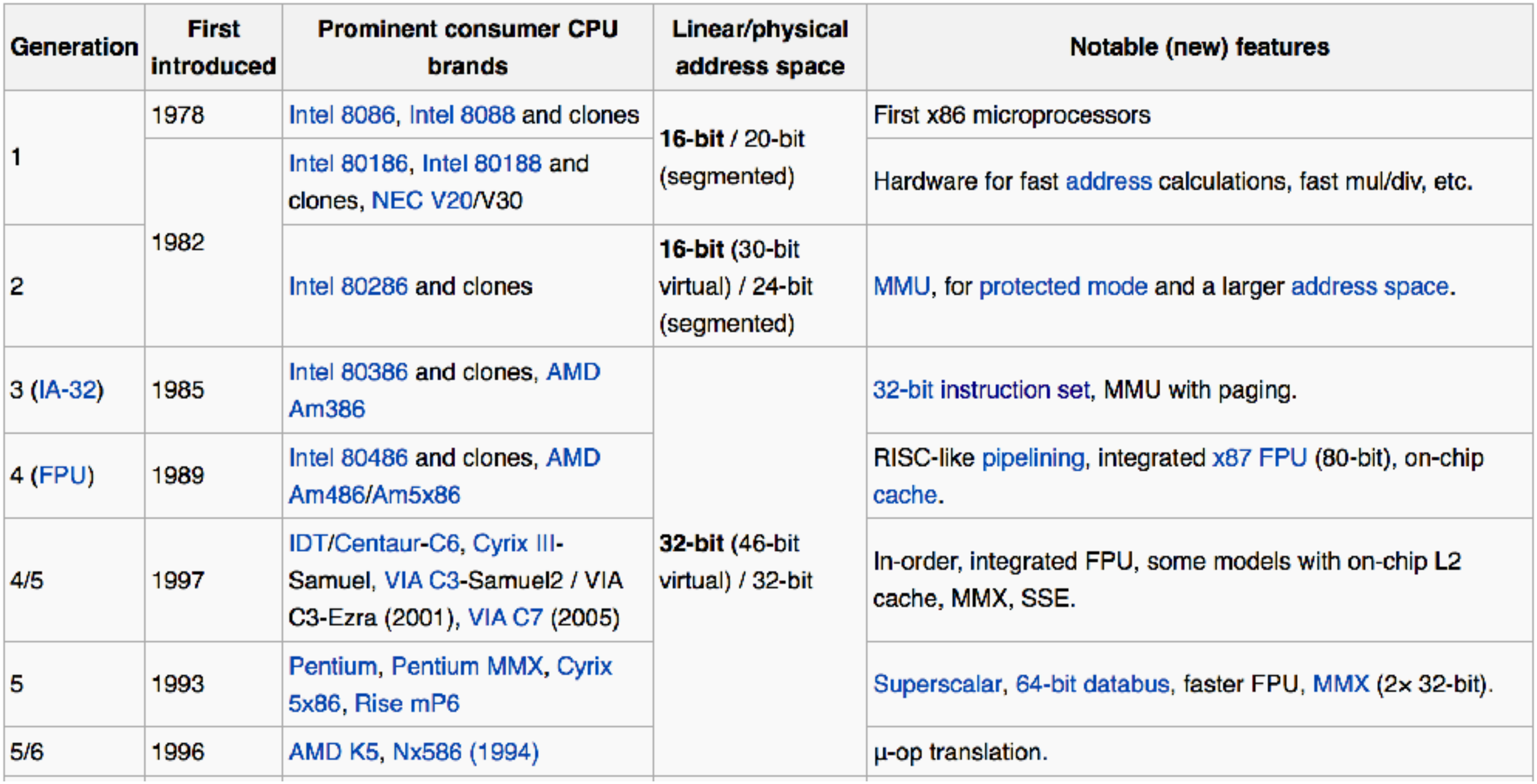
x86指令集

- x86架构于1978年推出的Intel 8086中央处理器中首度出现，它是从Intel 8008处理器中发展而来的，而8008则是发展自Intel 4004的。8086在三年后为IBM PC所选用，之后x86便成为了个人计算机的标准平台，是目前世界上最流行的台式机体系结构。

x86指令集的发展历史

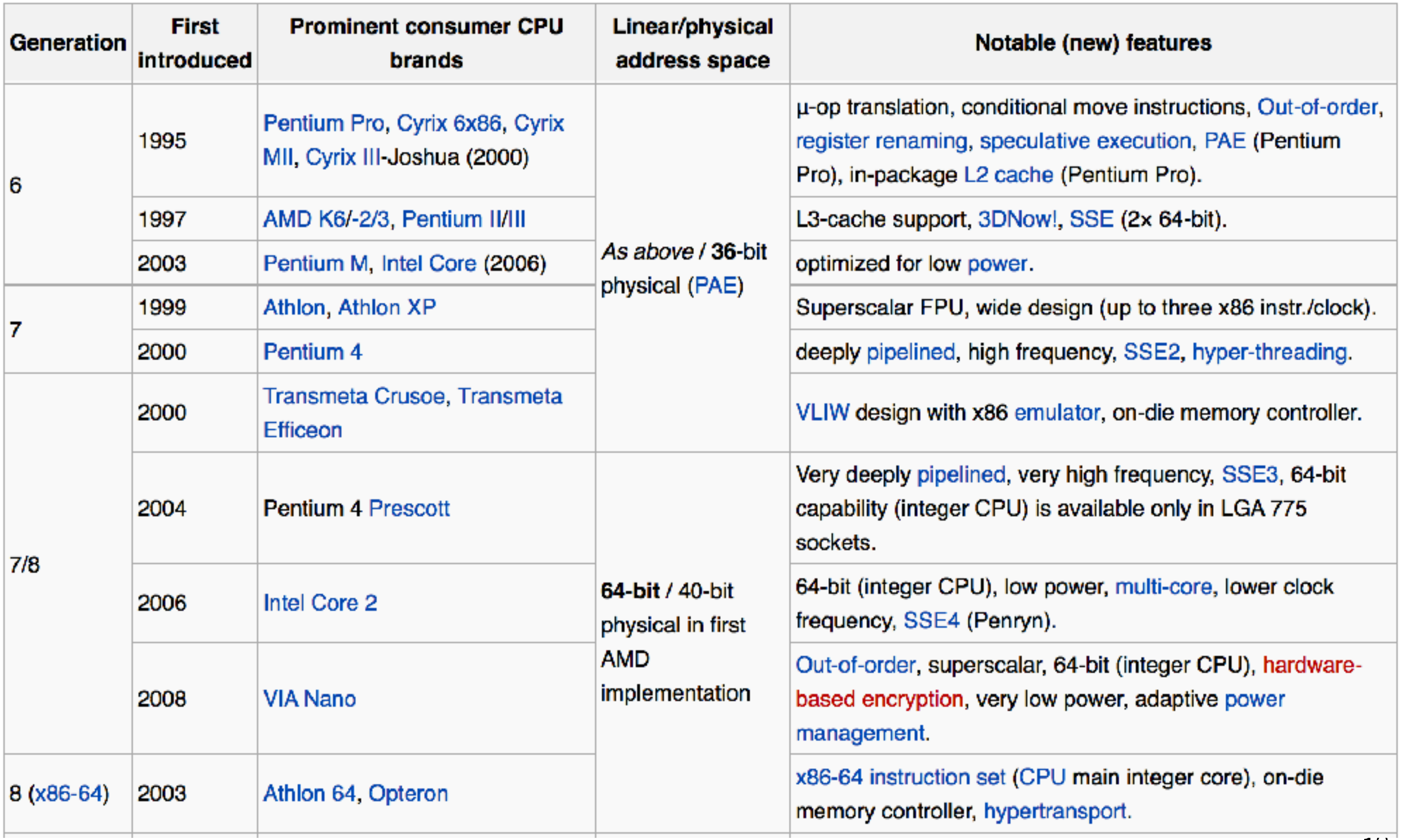
- 1978：Intel 8086，16位的体系结构
- 1980：Intel 8087浮点协处理器发布，8086基础上增加60条浮点指令
- 1982：80286在8086基础上把地址空间扩展到24位
- 1985：80386在80286基础上将地址空间扩展到32位
- 1989：80486
- 1992：Pentium处理器
- 1995：Pentium Pro处理器

x86指令集的发展历史



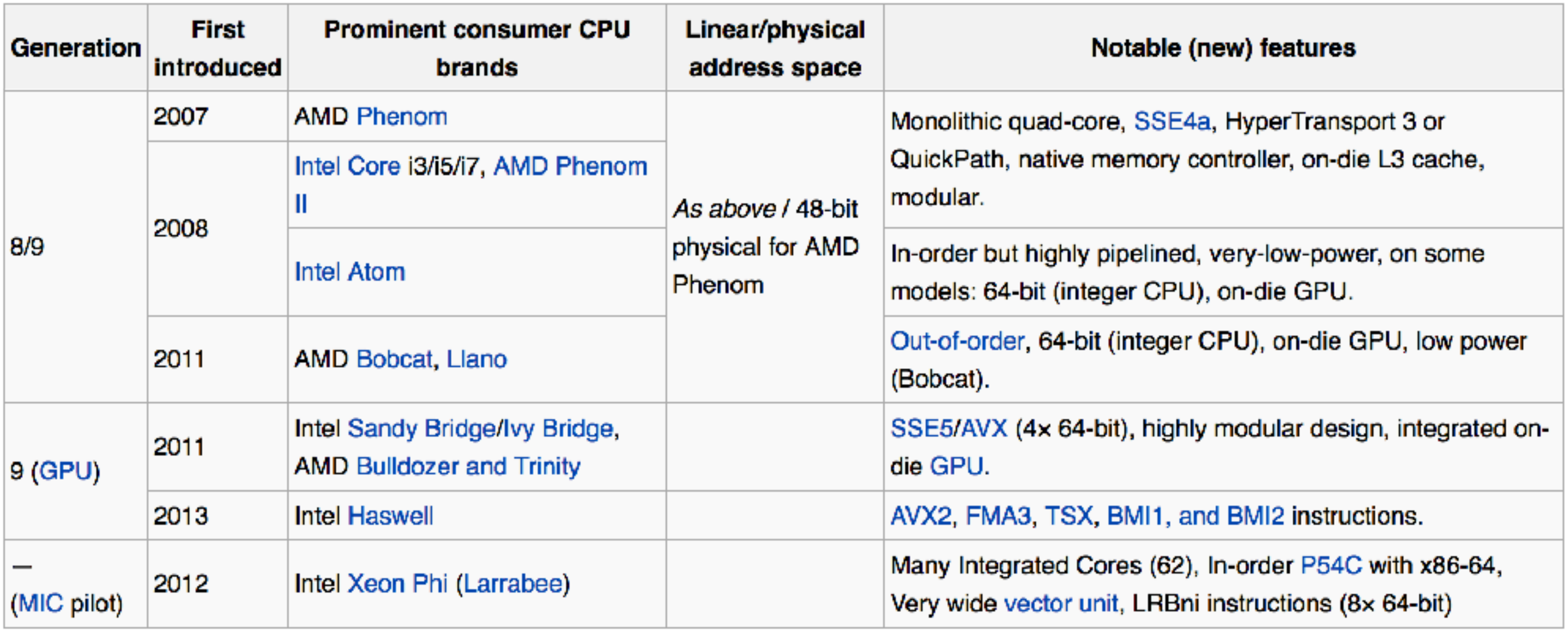
Generation	First introduced	Prominent consumer CPU brands	Linear/physical address space	Notable (new) features
1	1978	Intel 8086, Intel 8088 and clones	**16-bit** / 20-bit (segmented)	First x86 microprocessors
	1982	Intel 80186, Intel 80188 and clones, NEC V20/V30		Hardware for fast address calculations, fast mul/div, etc.
2		Intel 80286 and clones	**16-bit** (30-bit virtual) / 24-bit (segmented)	MMU, for protected mode and a larger address space.
3 (IA-32)	1985	Intel 80386 and clones, AMD Am386	**32-bit** (46-bit virtual) / 32-bit	32-bit instruction set, MMU with paging.
4 (FPU)	1989	Intel 80486 and clones, AMD Am486/Am5x86		RISC-like pipelining, integrated x87 FPU (80-bit), on-chip cache.
4/5	1997	IDT/Centaur-C6, Cyrix III-Samuel, VIA C3-Samuel2 / VIA C3-Ezra (2001), VIA C7 (2005)		In-order, integrated FPU, some models with on-chip L2 cache, MMX, SSE.
5	1993	Pentium, Pentium MMX, Cyrix 5x86, Rise mP6		Superscalar, 64-bit databus, faster FPU, MMX (2× 32-bit).
5/6	1996	AMD K5, Nx586 (1994)		μ-op translation.

x86指令集的发展历史



Generation	First introduced	Prominent consumer CPU brands	Linear/physical address space	Notable (new) features
6	1995	Pentium Pro, Cyrix 6x86, Cyrix MII, Cyrix III-Joshua (2000)	*As above* / **36**-bit physical (PAE)	µ-op translation, conditional move instructions, Out-of-order, register renaming, speculative execution, PAE (Pentium Pro), in-package L2 cache (Pentium Pro).
	1997	AMD K6/-2/3, Pentium II/III		L3-cache support, 3DNow!, SSE (2x 64-bit).
	2003	Pentium M, Intel Core (2006)		optimized for low power.
7	1999	Athlon, Athlon XP		Superscalar FPU, wide design (up to three x86 instr./clock).
	2000	Pentium 4		deeply pipelined, high frequency, SSE2, hyper-threading.
7/8	2000	Transmeta Crusoe, Transmeta Efficeon		VLIW design with x86 emulator, on-die memory controller.
	2004	Pentium 4 Prescott	**64-bit** / 40-bit physical in first AMD implementation	Very deeply pipelined, very high frequency, SSE3, 64-bit capability (integer CPU) is available only in LGA 775 sockets.
	2006	Intel Core 2		64-bit (integer CPU), low power, multi-core, lower clock frequency, SSE4 (Penryn).
	2008	VIA Nano		Out-of-order, superscalar, 64-bit (integer CPU), hardware-based encryption, very low power, adaptive power management.
8 (x86-64)	2003	Athlon 64, Opteron		x86-64 instruction set (CPU main integer core), on-die memory controller, hypertransport.

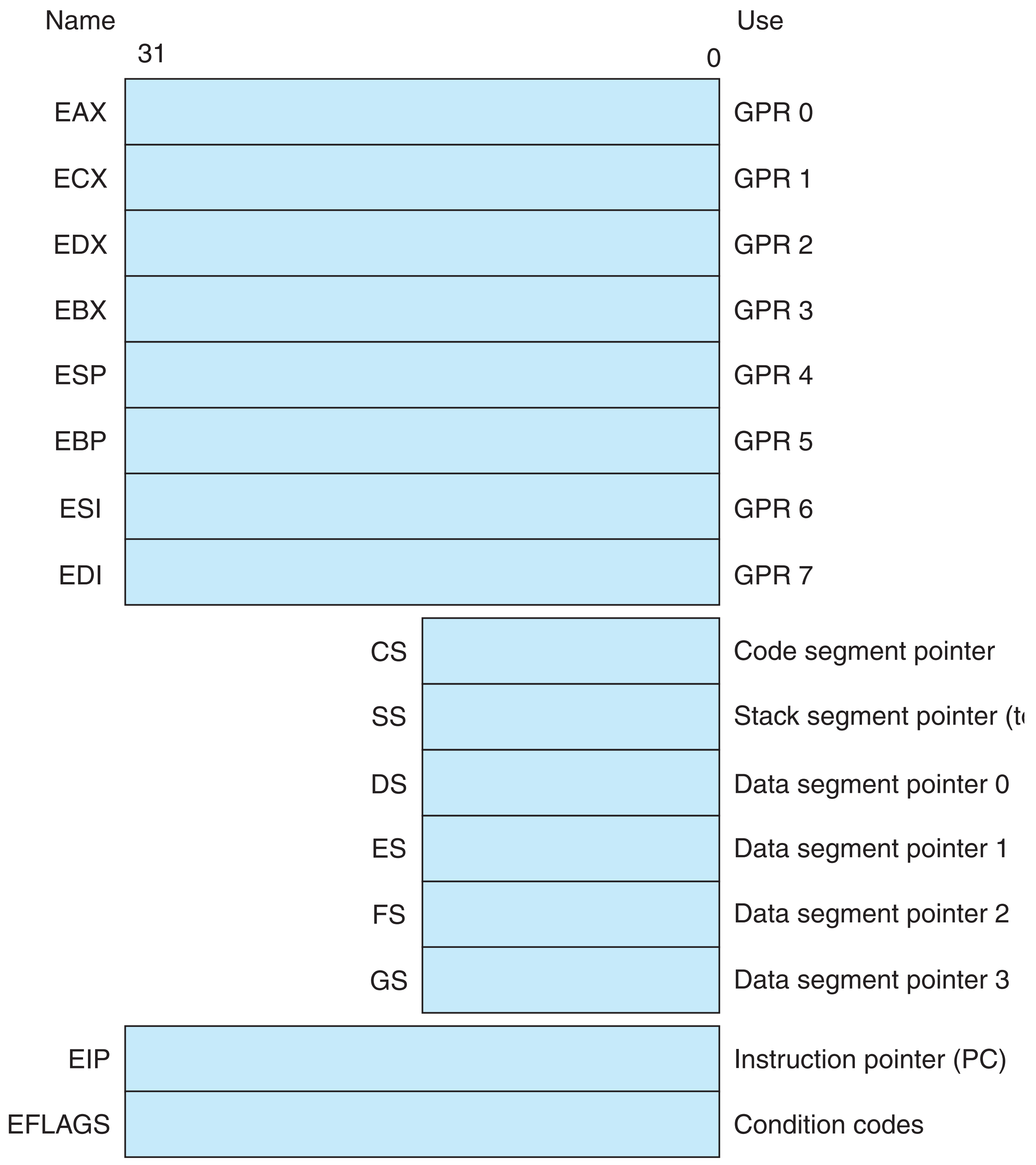
x86指令集的发展历史



Generation	First introduced	Prominent consumer CPU brands	Linear/physical address space	Notable (new) features
8/9	2007	AMD Phenom	*As above* / 48-bit physical for AMD Phenom	Monolithic quad-core, SSE4a, HyperTransport 3 or QuickPath, native memory controller, on-die L3 cache, modular.
	2008	Intel Core i3/i5/i7, AMD Phenom II		
		Intel Atom		In-order but highly pipelined, very-low-power, on some models: 64-bit (integer CPU), on-die GPU.
	2011	AMD Bobcat, Llano		Out-of-order, 64-bit (integer CPU), on-die GPU, low power (Bobcat).
9 (GPU)	2011	Intel Sandy Bridge/Ivy Bridge, AMD Bulldozer and Trinity		SSE5/AVX (4× 64-bit), highly modular design, integrated on-die GPU.
	2013	Intel Haswell		AVX2, FMA3, TSX, BMI1, and BMI2 instructions.
— (MIC pilot)	2012	Intel Xeon Phi (Larrabee)		Many Integrated Cores (62), In-order P54C with x86-64, Very wide vector unit, LRBni instructions (8× 64-bit)

x86寄存器和数据寻址模式

- 80386寄存器组
  - 80386把16位寄存器扩展为32位，并用前缀E标示，为通用寄存器
  - 8个通用寄存器



x86寄存器和数据寻址模式

- x86算术和逻辑指令中的一个操作数必须既是源操作数又是目的操作数
- x86一个操作数可以在存储器中

Source/destination operand type	Second source operand
Register	Register
Register	Immediate
Register	Memory
Memory	Register
Memory	Immediate

x86允许的操作数组合

三类基本指令

Transfer data between memory and register

- ***Load*** data from memory into register
  - %reg = Mem[address]
- ***Store*** register data into memory
  - Mem[address] = %reg

Remember: memory is indexed just like an array[] of bytes!

Perform arithmetic function on register or memory data

c = a + b;   z = x << y;   i = h & g;

Transfer control: what instruction to execute next

- Unconditional jumps to/from procedures
- Conditional branches

x86指令

Moving Data: IA32

Moving Data

`movx` ***Source, Dest***

`x` is one of {**`b, w, l`**}

`movl` ***Source, Dest*****:**

Move 4-byte "long word"

`movw` ***Source, Dest*****:**

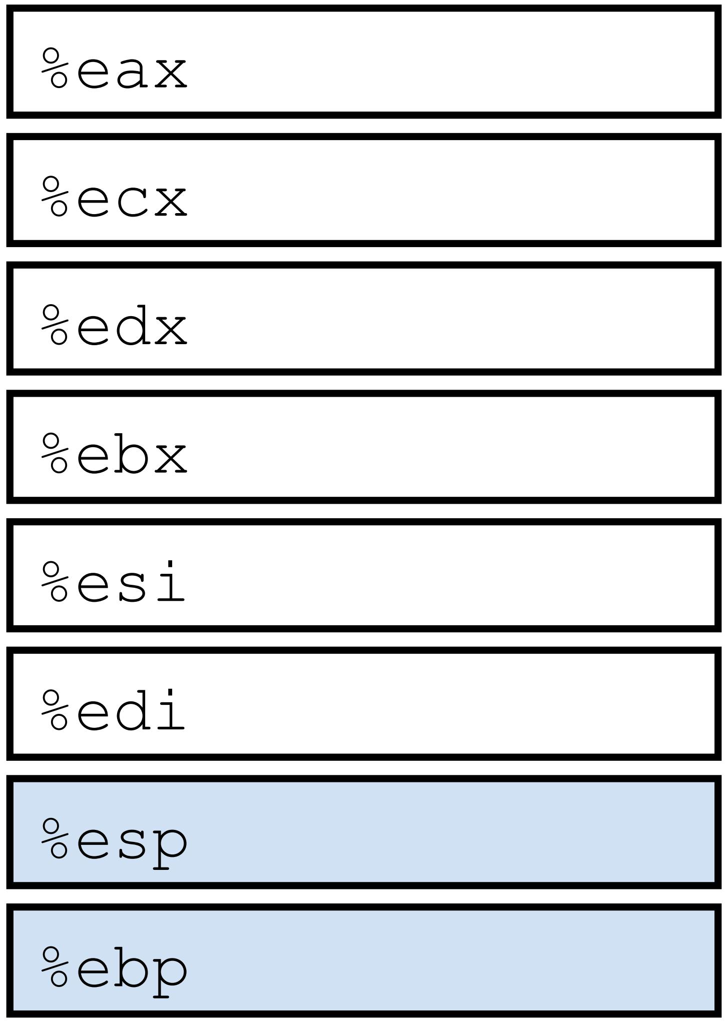
Move 2-byte "word"

`movb` ***Source, Dest*****:**

Move 1-byte "byte"

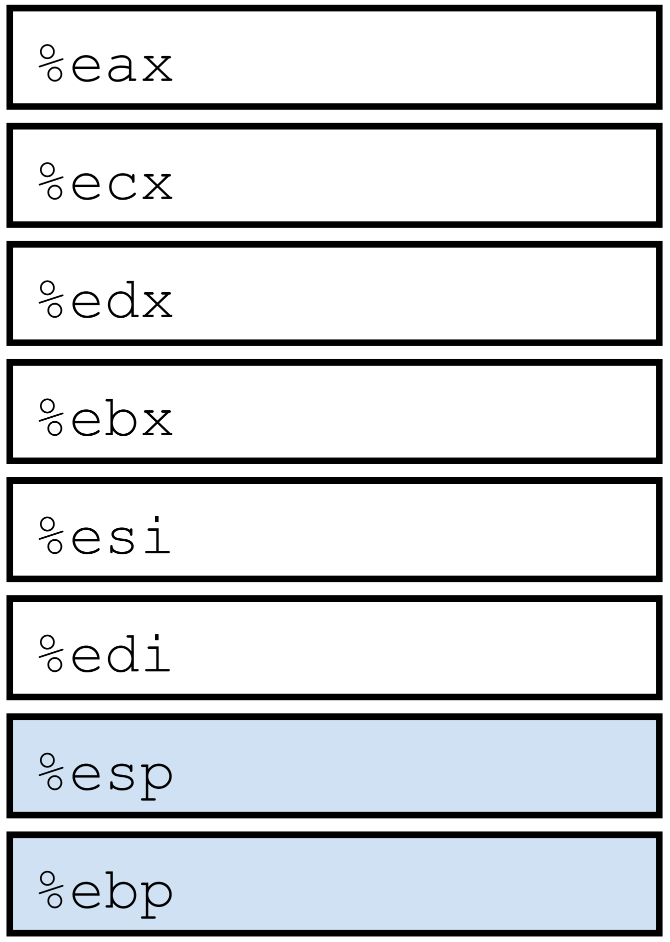
historical terms from the 16-bit days
not the current machine word size

Lots of these in typical code



x86指令

Moving Data: IA32



Moving Data

`movl` ***Source*, *Dest***:

Operand Types

Immediate: Constant integer data

Example: **`$0x400`, `$-533`**

Like C constant, but prefixed with **`'$'`**

Encoded with 1, 2, or 4 bytes

Register: One of 8 integer registers

Example: **`%eax`, `%edx`**

But **`%esp`** and **`%ebp`** reserved for special use

Others have special uses for particular instructions

Memory: 4 consecutive bytes of memory at address given by register

Simplest example: **`(%eax)`**

Various other “address modes”

x86指令

`movl` Operand Combinations

	Source	Dest	Src,Dest	C Analog
`movl`	*Imm*	*Reg*	`movl $0x4,%eax`	`var_a = 0x4;`
		Mem	`movl $-147,(%eax)`	`*p_a = -147;`
	Reg	*Reg*	`movl %eax,%edx`	`var_d = var_a;`
		Mem	`movl %eax,(%edx)`	`*p_d = var_a;`
	Mem	*Reg*	`movl (%eax),%edx`	`var_d = *p_a;`

Cannot do memory-memory transfer with a single instruction.

x86指令

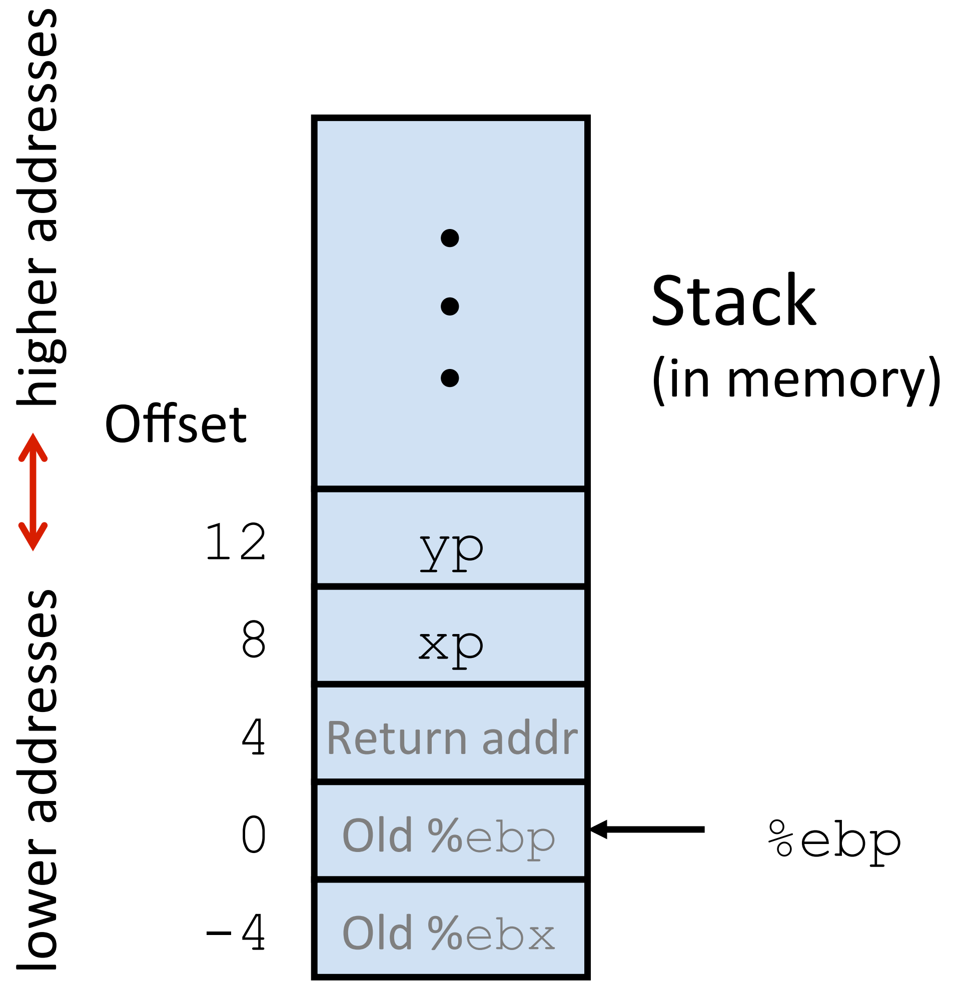
```c
void swap(int *xp, int *yp)
{
  int t0 = *xp;
  int t1 = *yp;
  *xp = t1;
  *yp = t0;
}
```

```
swap:
   pushl %ebp                  } Set
   movl  %esp,%ebp             } Up
   pushl %ebx

   movl 12(%ebp),%ecx          }
   movl 8(%ebp),%edx           }
   movl (%ecx),%eax            } Body
   movl (%edx),%ebx            }
   movl %eax,(%edx)            }
   movl %ebx,(%ecx)            }

   movl -4(%ebp),%ebx          }
   movl %ebp,%esp              } Finish
   popl %ebp                   }
   ret
```

x86指令

Understanding Swap

```
void swap(int *xp, int *yp)
{
  int t0 = *xp;
  int t1 = *yp;
  *xp = t1;
  *yp = t0;
}
```



Register	Value
%ecx	yp
%edx	xp
%eax	t1
%ebx	t0

register <-> variable mapping

```
movl 12(%ebp),%ecx   # ecx = yp
movl 8(%ebp),%edx    # edx = xp
movl (%ecx),%eax     # eax = *yp (t1)
movl (%edx),%ebx     # ebx = *xp (t0)
movl %eax,(%edx)     # *xp = eax
movl %ebx,(%ecx)     # *yp = ebx
```

x86寄存器和数据寻址模式

Mode	Description	Register restrictions	MIPS equivalent
Register indirect	Address is in a register.	not ESP or EBP	`lw $s0,0($s1)`
Based mode with 8- or 32-bit displacement	Address is contents of base register plus displacement.	not ESP or EBP	`lw $s0,100($s1) # 16-bit # displacement`
Base plus scaled index	The address is Base + (2^{Scale} x Index) where Scale has the value 0, 1, 2, or 3.	Base: any GPR Index: not ESP	`mul $t0,$s2,4` `add $t0,$t0,$s1` `lw $s0,0($t0)`
Base plus scaled index with 8- or 32-bit displacement	The address is Base + (2^{Scale} x Index) + displacement where Scale has the value 0, 1, 2, or 3.	Base: any GPR Index: not ESP	`mul $t0,$s2,4` `add $t0,$t0,$s1` `lw $s0,100($t0) # 16-bit # displacement`

- 每种寻址模式对于使用哪些寄存器是有限制的

x86的整数操作

- 四类整数操作
  - 数据传输指令
  - 算术和逻辑指令
  - 控制流
  - 字符串指令

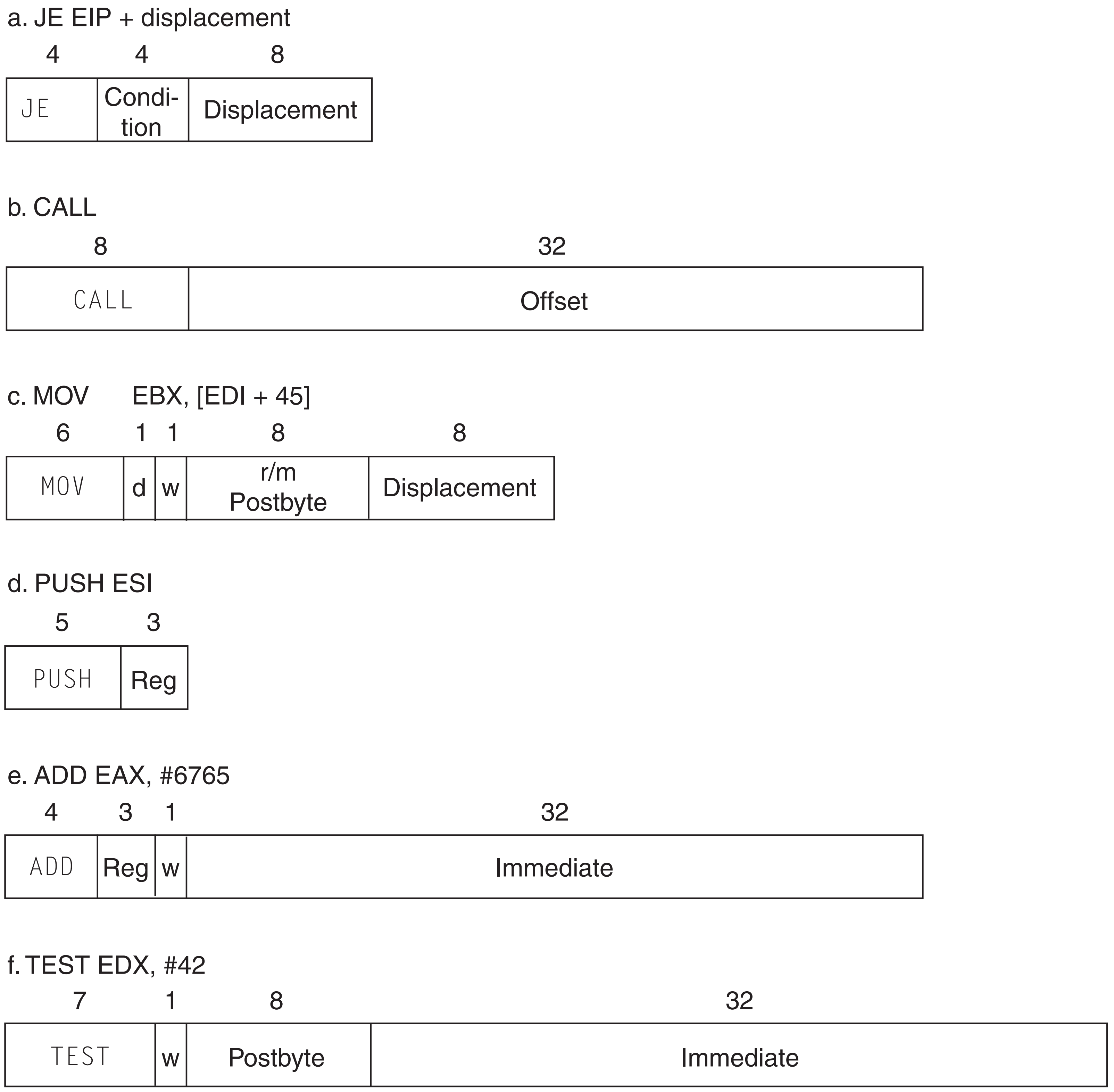
Instruction	Function
JE name	if equal(condition code) {EIP=name}; EIP-128 name < EIP+128
JMP name	EIP=name
CALL name	SP=SP-4; M[SP]=EIP+5; EIP=name;
MOVW EBX,[EDI+45]	EBX=M[EDI+45]
PUSH ESI	SP=SP-4; M[SP]=ESI
POP EDI	EDI=M[SP]; SP=SP+4
ADD EAX,#6765	EAX= EAX+6765
TEST EDX,#42	Set condition code (flags) with EDX and 42
MOVSL	M[EDI]=M[ESI]; EDI=EDI+4; ESI=ESI+4

典型的x86操作

Instruction	Meaning
Control	**Conditional and unconditional branches**
JNZ, JZ	Jump if condition to EIP + 8-bit offset; JNE (for JNZ), JE (for JZ) are alternative names
JMP	Unconditional jump—8-bit or 16-bit offset
CALL	Subroutine call—16-bit offset; return address pushed onto stack
RET	Pops return address from stack and jumps to it
LOOP	Loop branch—decrement ECX; jump to EIP + 8-bit displacement if ECX ≠ 0
Data transfer	**Move data between registers or between register and memory**
MOV	Move between two registers or between register and memory
PUSH, POP	Push source operand on stack; pop operand from stack top to a register
LES	Load ES and one of the GPRs from memory
Arithmetic, logical	**Arithmetic and logical operations using the data registers and memory**
ADD, SUB	Add source to destination; subtract source from destination; register-memory format
CMP	Compare source and destination; register-memory format
SHL, SHR, RCR	Shift left; shift logical right; rotate right with carry condition code as fill
CBW	Convert byte in 8 rightmost bits of EAX to 16-bit word in right of EAX
TEST	Logical AND of source and destination sets condition codes
INC, DEC	Increment destination, decrement destination
OR, XOR	Logical OR; exclusive OR; register-memory format
String	**Move between string operands; length given by a repeat prefix**
MOVS	Copies from string source to destination by incrementing ESI and EDI; may be repeated
LODS	Loads a byte, word, or double word of a string into the EAX register

x86指令编码

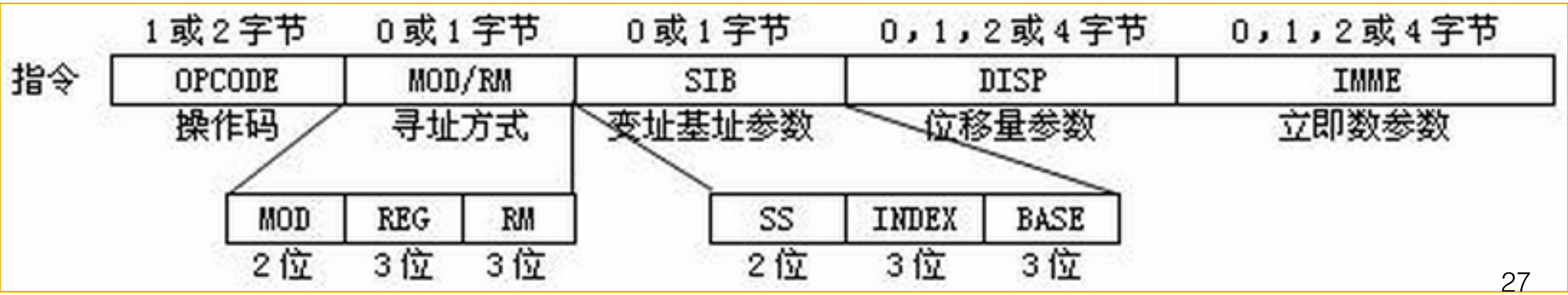
- 指令编码非常复杂
- 最短1字节，最长15字节



典型的x86指令格式

Pentium处理器的寻址方式

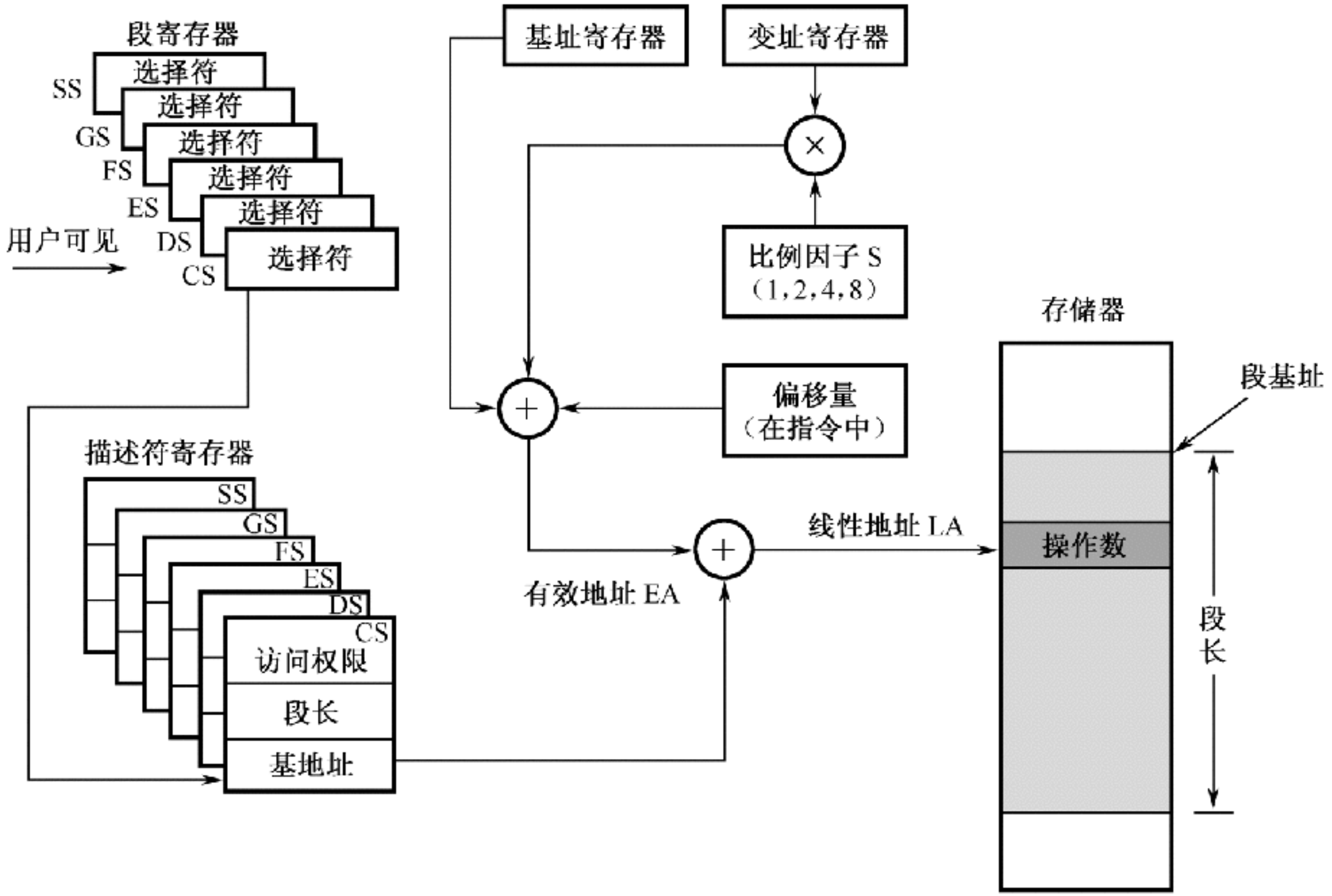
- 内存实际地址由两部分组成：存储单元所在段的基地址/段内偏移地址（偏移量）
- 段内偏移地址可以由如下四个部分组成（称为偏移地址四元素）：
  - 基址寄存器内容
  - 变址寄存器内容
  - 比例因子
  - 位移量



Pentium处理器的寻址方式

- 由四元素组合形成的偏移地址称为有效地址EA:
  - EA=基址+(变址×比例因子)+位移量
- 对于实模式（16位寻址）：
  - 基址寄存器：BX,BP
  - 变址寄存器：SI,DI
  - 比例因子： 0,1
  - 位移量： 0,8,16位
- 对于保护模式（32位寻址）：
  - 基址寄存器：任何32位通用寄存器
  - 变址寄存器：除ESP外的任何32位通用寄存器
  - 比例因子： 1,2,4,8
  - 位移量： 0,8,32位

Pentium处理器的寻址方式

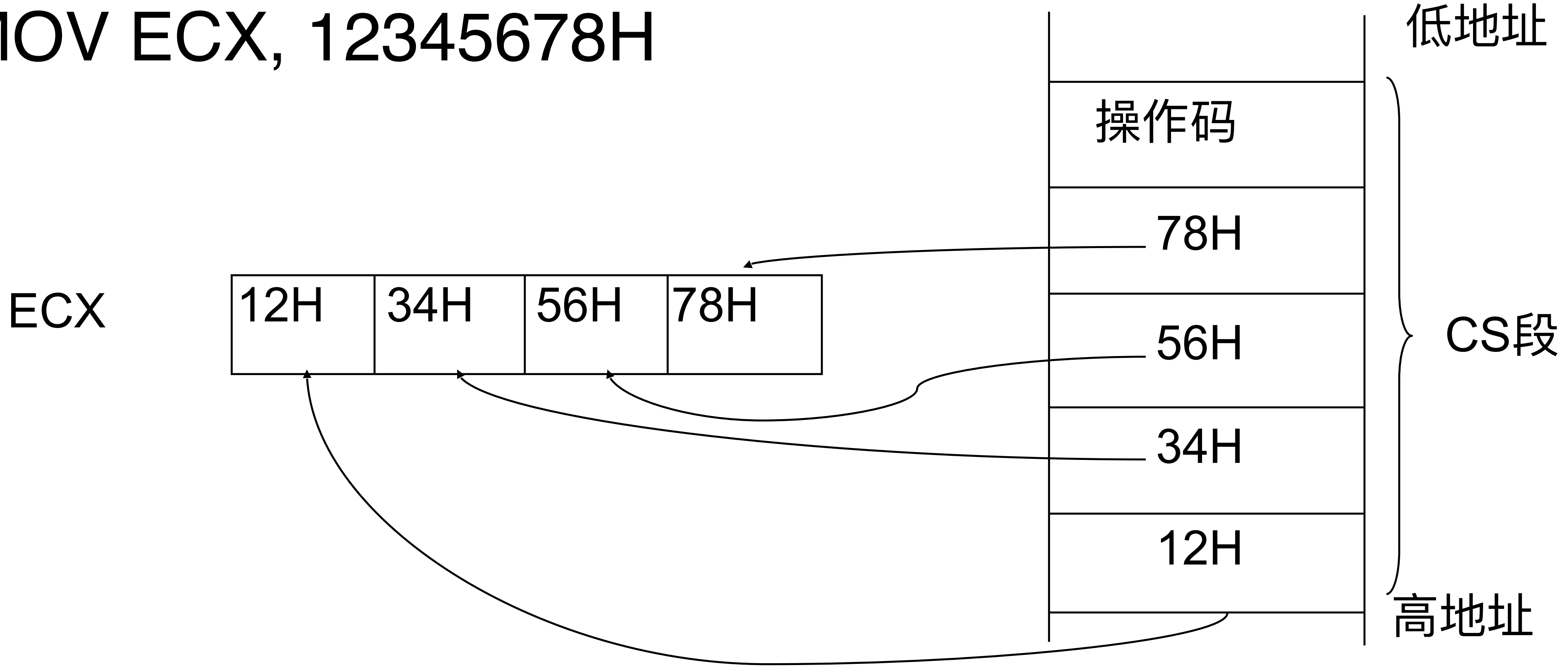


Pentium处理器的寻址方式

方式	算法
立即	作数=A
寄存器	EA=R
偏移量	EA=(SR)+A
基址	EA=(SR)+(B)
基址带偏移量	EA=(SR)+(B)+A
比例变址带偏移量	EA=(SR)+(I)×S+A
基址带变址和偏移量	EA=(SR)+(B)+(I)+A
基址带比例变址和偏移量	EA=(SR)+(B)+(I)×S+A
相对	EA=(PC)+A

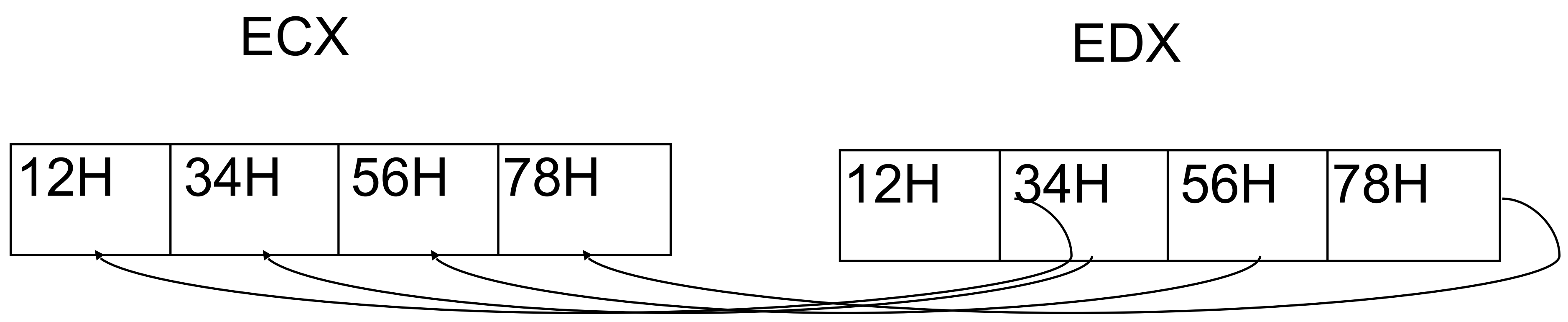
Pentium寻址方式

- 由四元素可组合出9种存储器寻址方式。
  - Pentium微处理器共有11种寻址方式.
- （1）立即数寻址
  - 操作数作为立即数直接存在指令中，可为字节、字、双字
  - MOV ECX, 12345678H



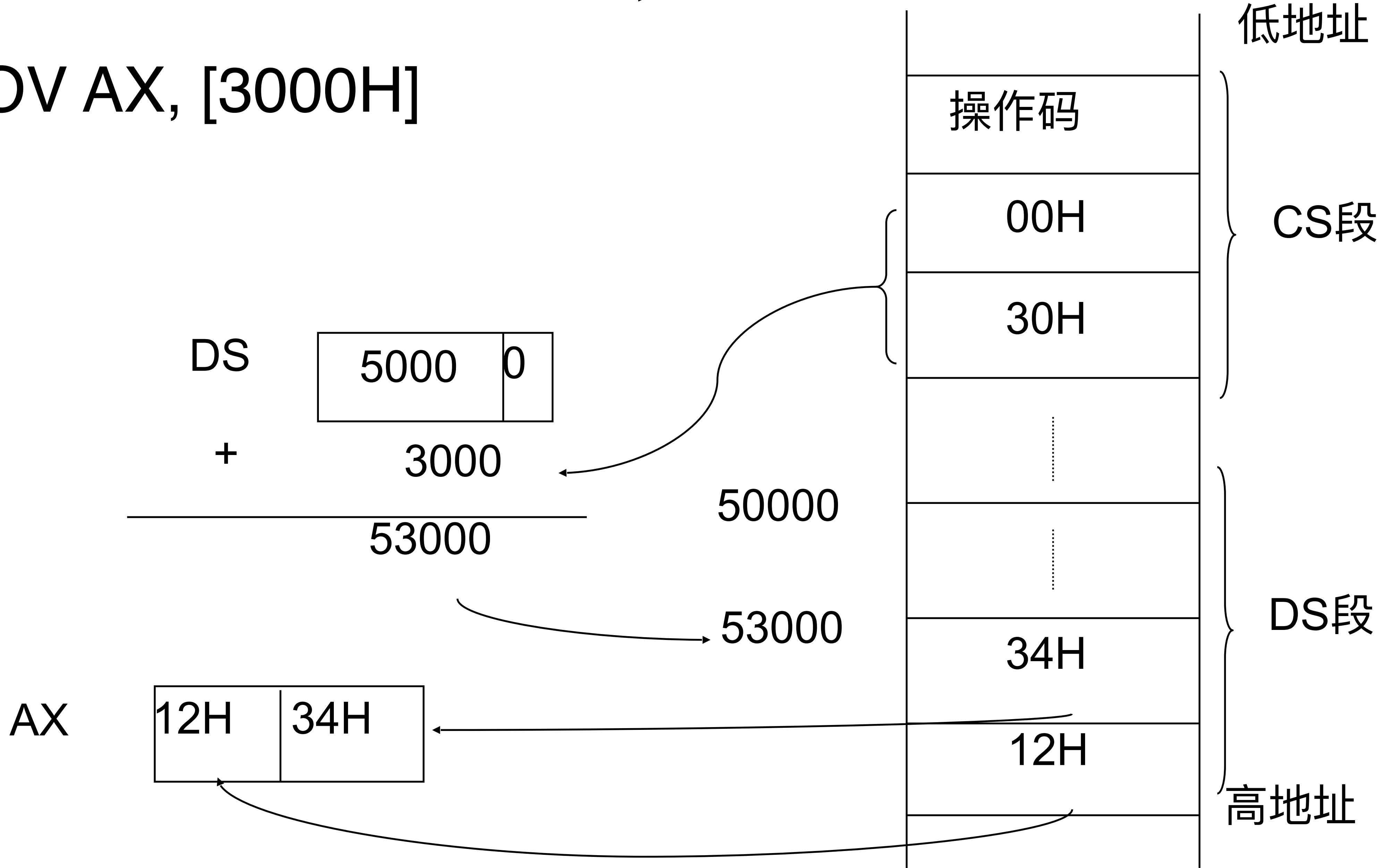
Pentium寻址方式

- （2）寄存器寻址
  - 操作数包含在指令规定的8位、16位、32位寄存器中
  - MOV ECX, EDX
  - 寄存器寻址由于无需从存储器中取操作数，故执行速度快



Pentium寻址方式

- （3）直接寻址
  - 指令中的操作数部分直接给出操作数的有效地址EA，操作数可以是16位或32位整数，操作数默认在DS段中
  - MOV AX, [3000H]

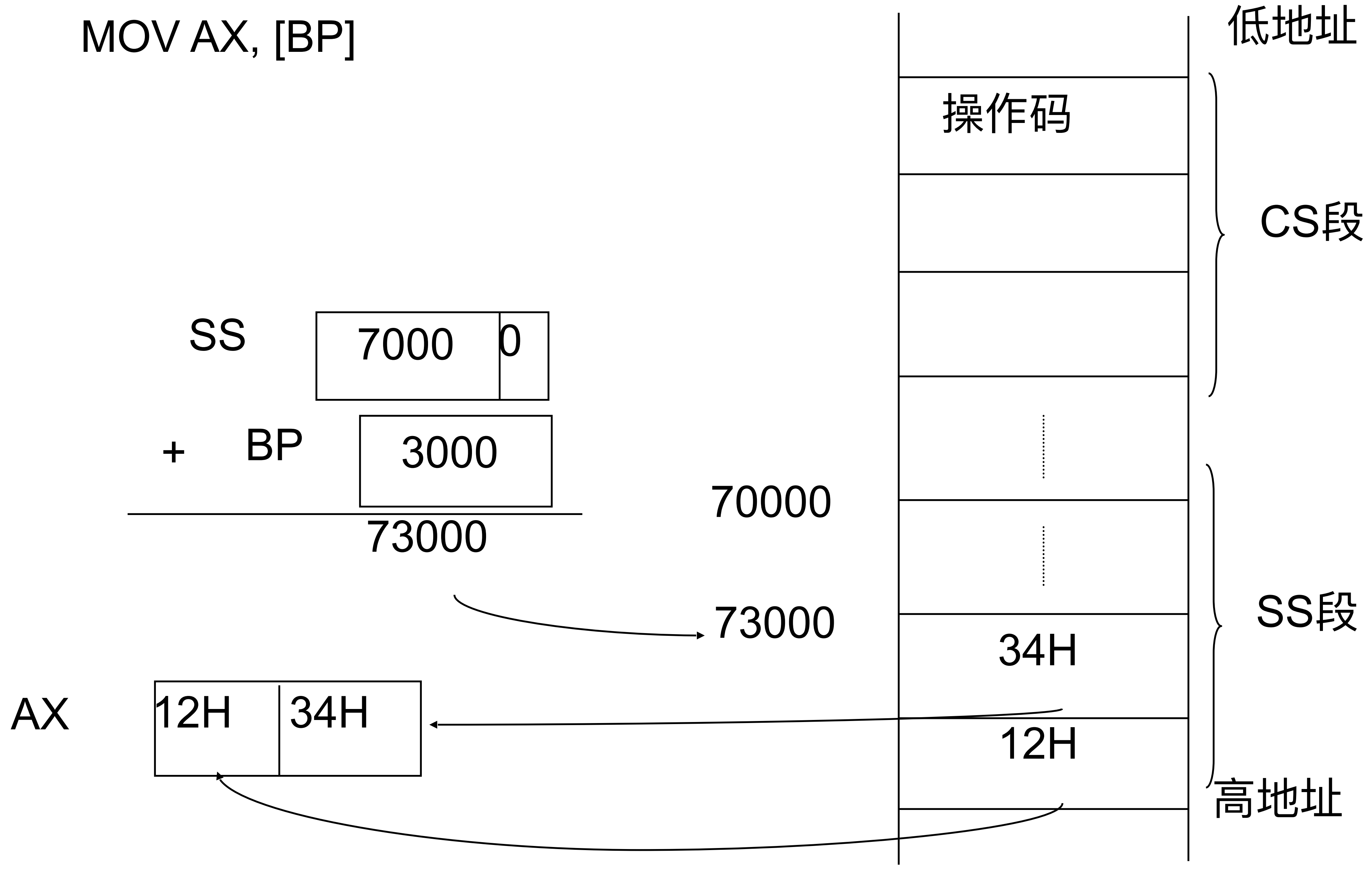


Pentium寻址方式

- （4）寄存器间接寻址
  - 操作数地址的偏移量（有效地址EA）存放在寄存器中
  - 16位寻址：偏移地址放在SI,DI,BP,BX中
    - 以SI,DI, BX间接寻址，默认操作数在DS段中
    - MOV AX, [SI]
    - 以BP间接寻址，默认操作数在SS段中
    - MOV AX, [BP]
  - 32位寻址：偏移地址放在8个32位通用寄存器中
    - 除ESP,EBP默认段寄存器为SS外，其余均默认段寄存器为DS
    - MOV EAX, [ESP]

Pentium寻址方式

- （4）寄存器间接寻址



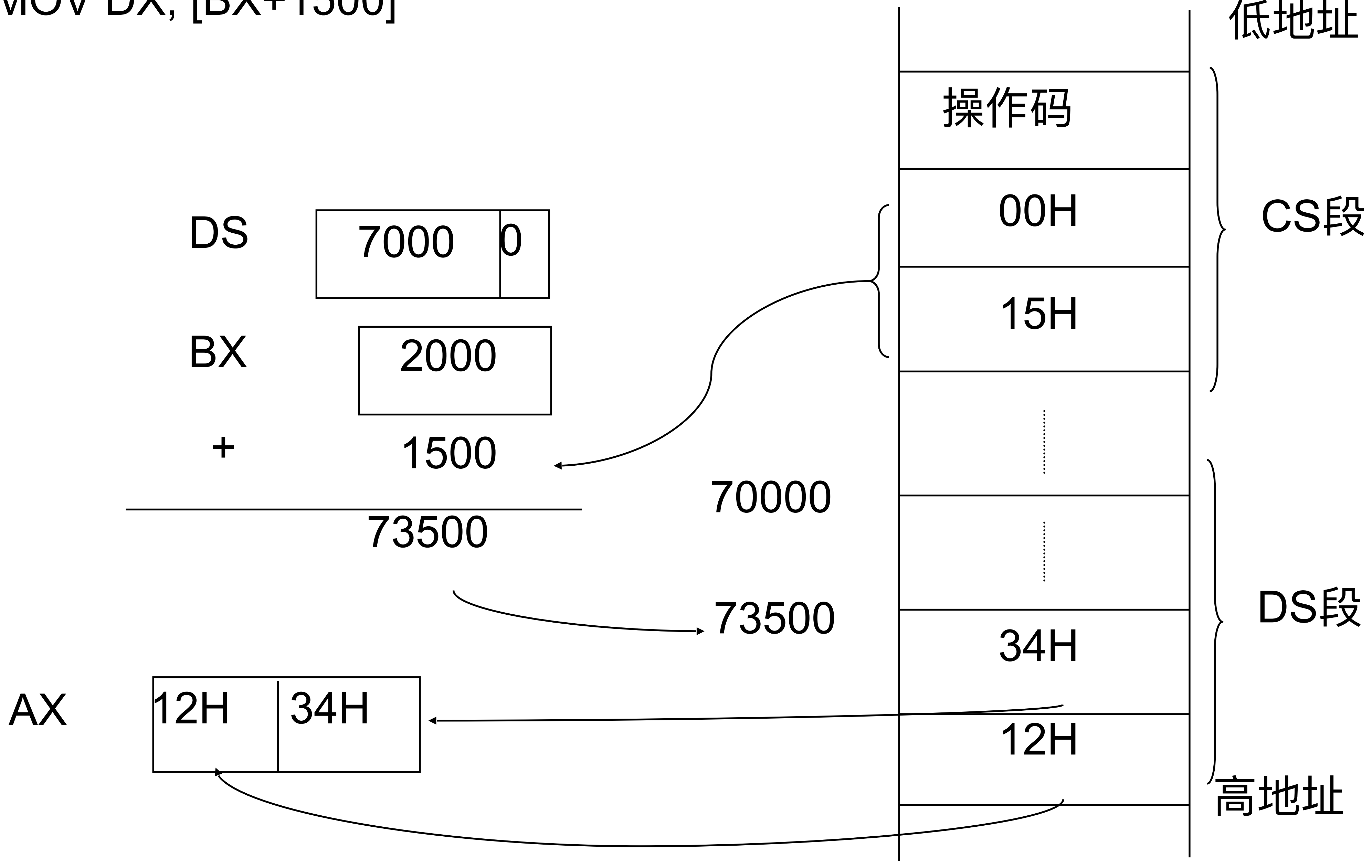
Pentium寻址方式

- （5）基址寻址
  - EA=[基址寄存器]+位移量
  - 16位寻址：BP,BX为基址寄存器
    - BX，DS为默认段寄存器
    - BP，SS为默认段寄存器
  - 32位寻址：8个32位通用寄存器均可作为基址寄存器，除ESP,EBP默认段寄存器为SS外，其余均默认段寄存器为DS
  - MOV EAX, [BX+24]
  - MOV DX, [EAX+1500]

Pentium寻址方式

- （5）基址寻址

MOV DX, [BX+1500]



Pentium寻址方式

- （6）变址寻址
  - EA=[变址寄存器]+位移量
  - 16位寻址：
    - SI,DI为基址寄存器, DS为默认段寄存器
  - 32位寻址：
    - 除ESP外其余7个32位通用寄存器均可作为变址寄存器，EBP默认SS为段寄存器，其余均默认段寄存器为DS
  - MOV AH, [SI+5]
  - 变址寻址适用于对一维数组的元素进行操作。

Pentium寻址方式

- （7）比例变址寻址
  - EA=[变址寄存器]×比例因子+位移量
  - 只适用于32位寻址
  - MOV EAX, [ESI*4+50]
  - 比例变址寻址适用于一维数组操作，当数组元素大小为2/4/8字节时，它更方便、有效

Pentium寻址方式

- （8）基址加变址寻址
  - EA=[基址寄存器]+[变址寄存器]
  - 适用于16位和32位寻址
  - MOV AX, [BX+SI]
  - MOV EAX, [EDX+EBP]
  - 基址加变址寻址主要用于二维数组操作和二重循环

Pentium寻址方式

- （9）基址加比例变址寻址

  - EA=[变址寄存器] ×比例因子+[基址寄存器]
  - 只适用于32位寻址

  - MOV EAX, [EDX*8+EAX]
  - 适用于数组元素大小为2/4/8字节时二维数组操作

Pentium寻址方式

- （10）带位移的基址加变址寻址
  - EA=[基址寄存器]+[变址寄存器]+位移量
  - 适用于16位和32位寻址
  - MOV AX, [BX+SI+50]
  - MOV EAX, [EDX+EBP+0FFFF000H]
  - 主要用于二维数组操作,位移量为数组起始地址

Pentium寻址方式

- （11）带位移的基址加比例变址寻址
  - EA=[变址寄存器] ×比例因子+[基址寄存器]+位移量
  - 只适用于32位寻址
  - MOV AX, [BX+SI+50]
  - MOV EAX, [EDX+EBP+0FFFF000H]
  - 适用于数组元素大小为2/4/8字节时二维数组操作,位移量为数组起始地址