COMP1001

Computer Systems

Dr. Vasilios Kelefouras

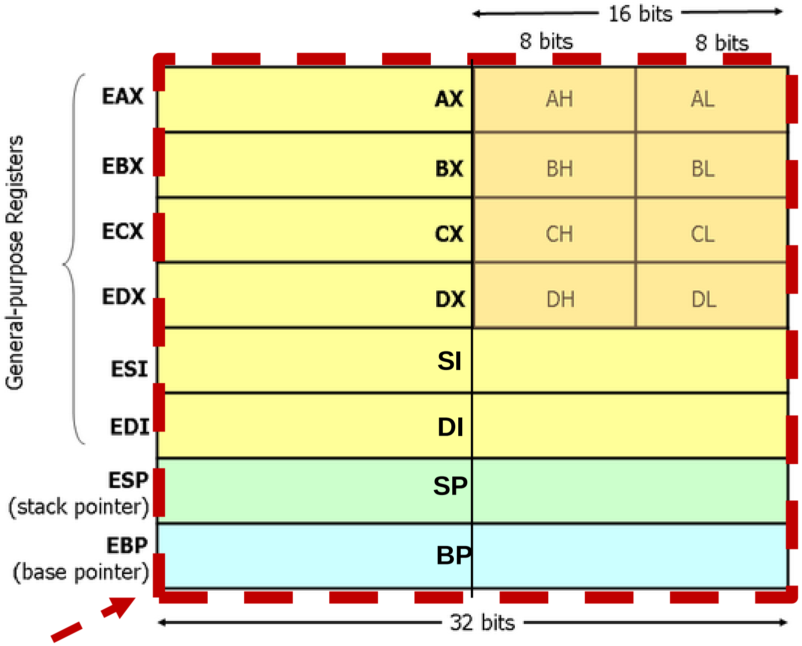
Email: v.kelefouras@plymouth.ac.uk
Website:
https://www.plymouth.ac.uk/staff/vasilios-kelefouras

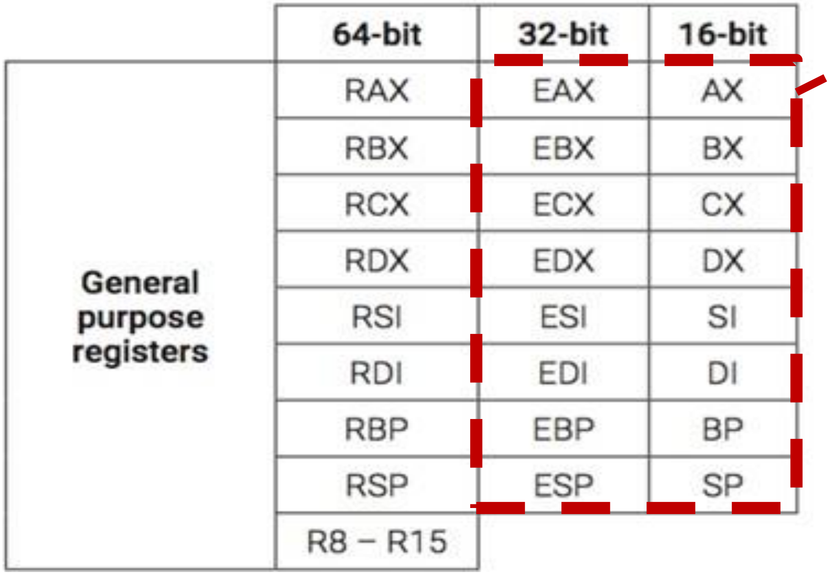
Outline

- Registers
- Multiplication
- Division

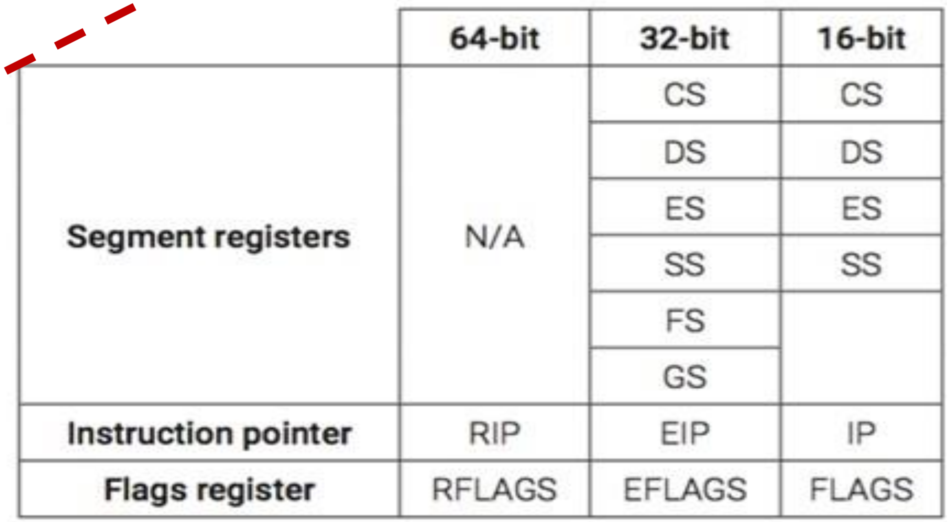
Registers (1)

- The lower bytes of some of these registers may be accessed independently as 32, 16 or 8-bit registers
- Older processors use 8bit, 16bit or 32bit registers only – compatibility exists
- There are other registers too…(next slide)





	64-bit	32-bit	16-bit
General purpose registers	RAX	EAX	AX
	RBX	EBX	BX
	RCX	ECX	CX
	RDX	EDX	DX
	RSI	ESI	SI
	RDI	EDI	DI
	RBP	EBP	BP
	RSP	ESP	SP
	R8 – R15		



	64-bit	32-bit	16-bit
Segment registers	N/A	CS	CS
		DS	DS
		ES	ES
		SS	SS
		FS	
		GS	
Instruction pointer	RIP	EIP	IP
Flags register	RFLAGS	EFLAGS	FLAGS

Registers (2)

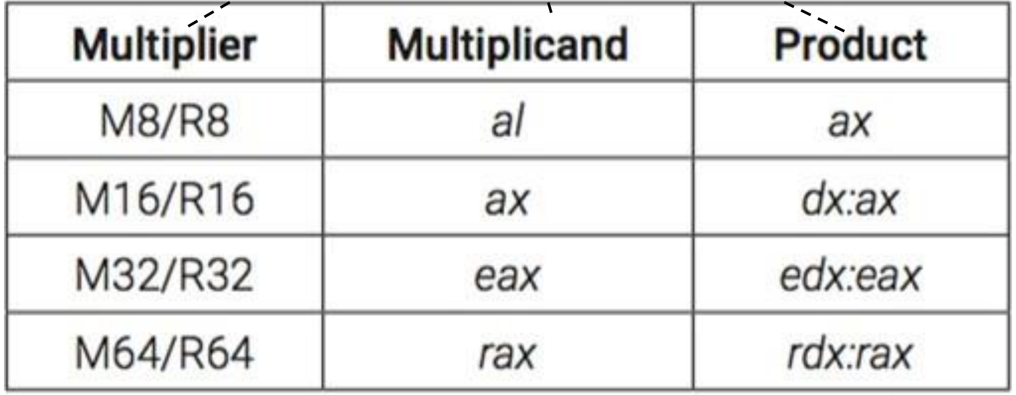
- **There are also eight 80bit floating point registers**
  - ST(0)-ST(7), arranged as a stack
- **Eight 64bit MMX vector registers**
  - Used with MMX instructions (physically they are the same as above)
- **Eight/Sixteen 128/256/512 bit vector registers**
  - 128bit use SSE instructions
  - 256bit use AVX instructions
  - 512bit use AVX2 instructions

Registers (3)

- ***rax/eax***: Default accumulator register.
  - Used for arithmetical operations
  - Function calls place return value.
  - Do not use it for data storage while performing such operations.
- ***rcx/ecx:*** Hold loop counter. Do not overwrite when looping!
- ***rbp/ebp***: Reference data on the stack; more on this later.
- ***rsp/esp***: Used for managing the stack – typically points to the top of the stack.
- ***rsi/esi*** and ***rdi/edi***: Index registers used in string operations.
- ***rip/eip:*** Instruction pointer - shows next instruction to be executed
- ***rflags/eflags***: Status and control registers; cannot be modified directly!

MUL (unsigned multiply)

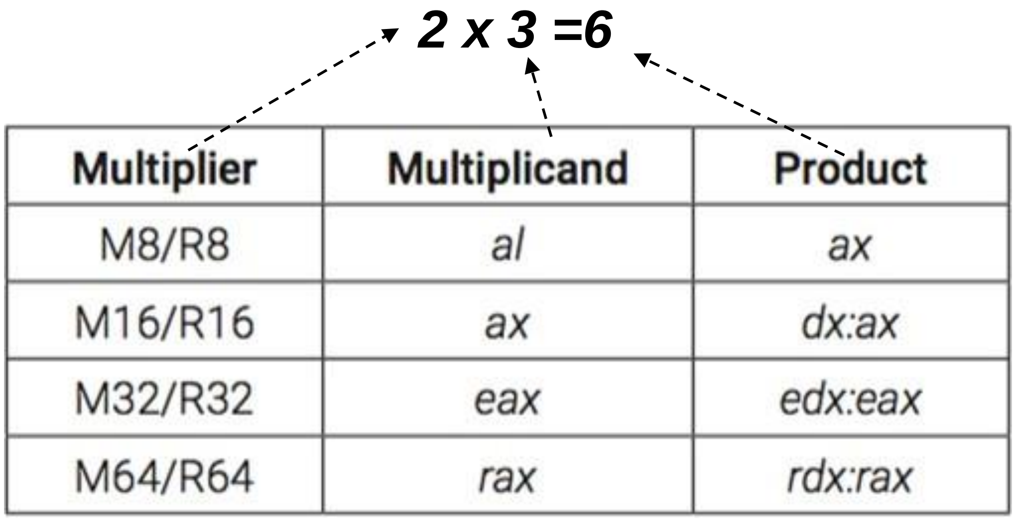
2 x 3 =6



Multiplier	Multiplicand	Product
M8/R8	*al*	*ax*
M16/R16	*ax*	*dx:ax*
M32/R32	*eax*	*edx:eax*
M64/R64	*rax*	*rdx:rax*

- □ Multiplication may require more bytes to hold the results. Consider the following 2-bit multiplicand 3_{10} (11_2) and 2-bit multiplier 3_{10} (11_2). The product is 9_{10} (1001_2), and it cannot be contained in 2-bits; it requires 4-bits. At most we require double the size of the multiplier or the multiplicand.
- □ Also, note that the parts of the product are saved in *high:low* format.

MUL - example



2 x 3 =6

Multiplier	Multiplicand	Product
M8/R8	al	ax
M16/R16	ax	dx:ax
M32/R32	eax	edx:eax
M64/R64	rax	rdx:rax

```
.data
var1  WORD 3000h
var2 WORD 100h

.code ; 16bit multiplication
mov ax,var1
mul var2 ; DX:AX = 00300000h, CF=1
```

CF=1 as DX contains non zero data

```
.data
var1 DWORD 3000h
var2 DWORD 100h

.code ; 32bit multiplication
mov eax,var1
mul var2 ; EDX:EAX = 0000000000300000h, CF=0
```

CF=0 as EDX is zero

IMUL – signed multiply

- ***imul*** is similar to ***mul***
- However:
  - It preserves the sign of the product by sign-extending it into the upper half of the destination register
  - It sets OF flag to ‘1’ when the less significant register cannot store the result (including its sign)

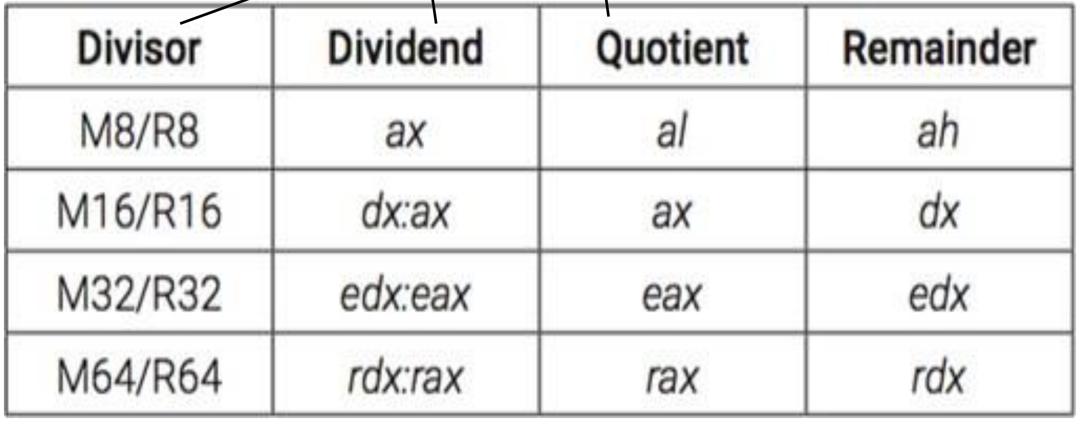
```
.data
var1 SBYTE 48 ; this is decimal
var2 SBYTE 4  ; this is decimal

.code ; 8bit multiplication
mov al,var1
imul var2 ; AH:AL = 00C0h, OF=1
```

OF=1 as 8bits are not enough to hold the signed number C0$_{16}$ (0 1100 0000$_2$). A ‘0’ is needed in AH to hold the sign

DIV (Unsigned Divide)

40 : 8 =5



Divisor	Dividend	Quotient	Remainder
M8/R8	ax	al	ah
M16/R16	dx:ax	ax	dx
M32/R32	edx:eax	eax	edx
M64/R64	rdx:rax	rax	rdx

```
.code ; 16bit division
 mov dx,0h          ; clear dividend, high
 mov ax,8003h  ; dividend, low
 mov cx,100h      ; divisor
 div cx              ; AX = 0080h, DX = 3
```

```
.code ; 32bit division
 mov edx,0 ; clear dividend, high
 mov eax,8003h ; dividend, low
 mov ecx,100h ; divisor
 div ecx ; EAX = 0000 0080h, EDX = 3
```

Any questions?

Further Reading

- Chapter 1 and Chapter 2 in ‘Modern X86 Assembly Language Programming’ , available at https://www.pdfdrive.com/download.pdf?id=185772000&h=3dfb070c1742f50b500f07a63a30c86a&u=cache&ext=pdf