

Digital microsystems design

Marius Marcu

2020

Objectives

- Specific objectives
 - Internal architecture and external interfaces of x86 processors and their functional behavior

Outline

- Definitions
- ISA classification
- x86 microprocessor internal architecture (microarchitecture)
- Instruction set architecture (ISA)
- Summary

Definitions

- Processor microarchitectures
 - Specifies the internal microprocessor architecture and the way instructions are implemented and executed
 - Includes:
 - Register file
 - Execution units
 - Pipeline
 - Internal caches
 - Does not include
 - Memory subsystem (except registers and internal caches)
 - I/O subsystem

Definitions

- Instruction set architecture (ISA)
 - Specifies the programmable operations set implemented by the processor and the way they can be used by programs
 - Includes:
 - Internal registers
 - Addressing modes
 - Instruction set
 - Memory model
 - Does not include
 - Memory subsystem (except registers)
 - I/O subsystem

Definitions

- Instruction set architecture
 - General vs. Specific
 - Specific: SSE, FP
 - Complete
 - Every operation supported by the processor should have at least one instruction that implements it – so that operations can be initiated from a program
 - Orthogonal
 - Addressing modes are independent of instruction type
 - The way operands are accessed are independent of the instructions using them

Classification

- Instruction set architecture
 - Compatibility
 - Data types
 - Data alignment
 - Addressing modes
 - Branches
 - Interrupts
 - Instruction encoding:
 - Fixed vs. variable size

Classification

- ISA types
 - CISC – Complex Instruction Set Computers
 - RISC – Reduced Instruction Set Computers
 - VLIW – Very Long Instruction Word
 - CMS – Code Morphic Software / Morphic processors

Classification

- CISC – Complex Instruction Set Computers
 - High performance can be achieved by reducing the number of instructions required to implement a program
 - Instructions' diversity and complex instructions reduce software size – less instructions to execute
 - They require complex microarchitectures and decoders to implement them
 - They provide a large set of both simple and complex instructions
 - Instructions are encoded on different number of bits

Classification

- RISC – Reduced Instruction Set Computers
 - High performance can be achieved by simplifying the instructions, and consequently by simplifying the microprocessor architecture
 - Simple instructions are fast and they can be implemented easily in hardware – fast execution of instructions
 - Few instructions will increase the software size
 - Processors provide a reduced set of simple instructions
 - Instructions are encoded on a fixed number of bits

Classification

- CISC vs. RISC

- Simple instructions vs. complex instructions:

- INC EBX

INC R1

- INC [EBX]

LD R1, [R2]

INC R1

LD [R2], R1

(CISC)

(RISC)

Classification

- CISC vs. RISC
 - CISC
 - Less instructions needed to implement a program
 - Smaller binary code of compiled programs
 - More flexible for programmers
 - Operands can be registers and memory locations

Classification

- CISC vs. RISC
 - RISC
 - Simple and faster instructions, more instruction executed per second
 - Easier and cheaper to be implemented
 - Dedicated instructions for memory access
 - Load/store instructions

Classification

- CISC vs. RISC
 - Add of two memory operands
 - The result is stored into memory

CISC

ADD (R3), (R2), (R1)

RISC

LOAD R4, (R1)

LOAD R5, (R2)

ADD R6, R4, R5

STORE (R3), R6

Classification

- CISC vs. RISC
 - RISC – the compiler optimizes the code
 - Static optimization
 - CISC – the microarchitecture optimizes the code
 - Dynamic optimization

Classification

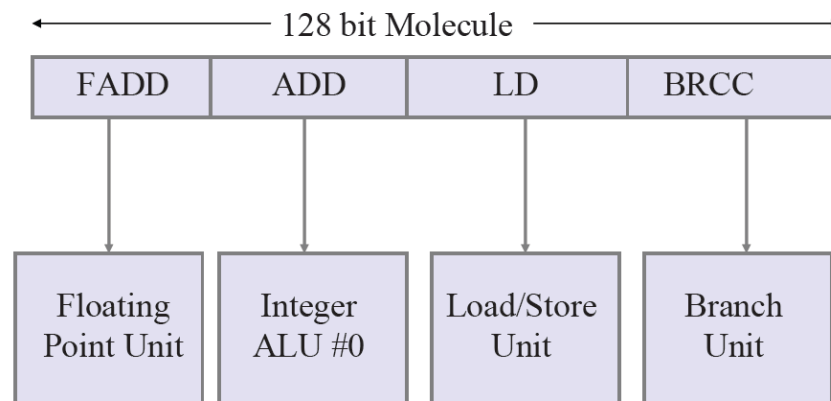
- CISC vs. RISC
 - It is difficult to say which architecture is better.
 - RISC architectures have taken useful instructions from CISC architectures and included them in their instruction sets.
 - CISC architectures have dropped instructions that were not useful (too complex).

Classification

- VLIW – Very Long Instruction Word
 - Several internal functional units operating in parallel
 - Instructions encoded on large number of bits, the same number of bits for every instruction
 - Compiler level ILP
 - Parallelism explicitly specified when compiling
 - The compiler aligns the machine code (instructions) selecting the instructions that can be executed in parallel
 - They have many internal general purpose registers

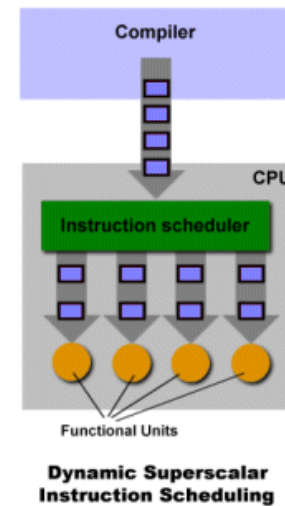
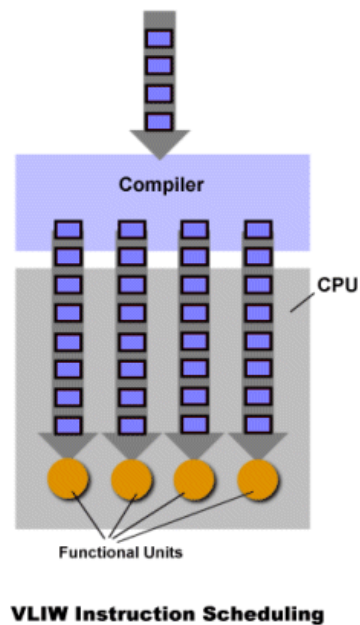
Classification

- VLIW
 - The generated code contains groups of instructions that will be executed simultaneously by the processor



Classification

- ILP: VLIW vs. CISC
 - Static
 - Dynamic


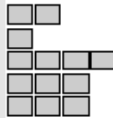
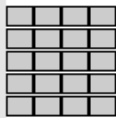
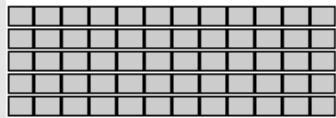


Classification

- VLIW
 - Compiling tools specific to an internal architecture
 - Changes to internal architecture will require changes to compiling toolset
 - Lack of compatibility between tools

Classification

- Summary

ARCHITECTURE CHARACTERISTIC	CISC	RISC	VLIW
INSTRUCTION SIZE	Varies	One size, usually 32 bits	One size
INSTRUCTION FORMAT	Field placement varies	Regular, consistent placement of fields	Regular, consistent placement of fields
INSTRUCTION SEMANTICS	Varies from simple to complex; possibly many dependent operations per instruction	Almost always one simple operation	Many simple, independent operations
REGISTERS	Few, sometimes special	Many, general-purpose	Many, general-purpose
MEMORY REFERENCES	Bundled with operations in many different types of instructions	Not bundled with operations, i.e., load/store architecture	Not bundled with operations, i.e., load/store architecture
HARDWARE DESIGN FOCUS	Exploit microcoded implementations	Exploit implementations with one pipeline and & no microcode	Exploit implementations with multiple pipelines, no microcode & no complex dispatch logic
PICTURE OF FIVE TYPICAL INSTRUCTIONS  = 1 BYTE			

Classification

- Summary

```
function (j)
long j; {
long i;

    j = j + i;
}
```

CISC

```
add 4[r1] <- r2
```

addMR	4	r1	r5
-------	---	----	----

RISC

```
load r5 <- 4[r1]
add r5 <- r5 + r2
store 4[r1] <- r5
```

load	r5	r1	4
add	r5	r5	r2
store	r5	r1	4

VLIW

```
load r5 <- 4[r1]
add r5 <- r5 + r2
store 4[r1] <- r5
```

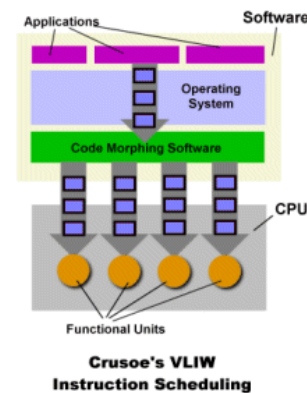
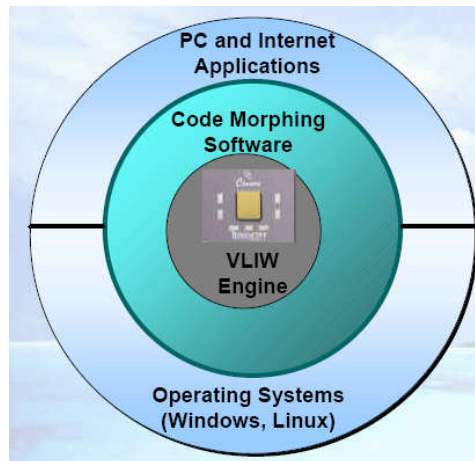
-	-	-	-	-	-	-	-	load	r5	r1	4
-	-	-	-	add	r5	r5	r2	-	-	-	-
-	-	-	-	-	-	-	-	store	r5	r1	4

Classification

- Morphic processors
 - Code Morphic Software
 - The processor architecture is VLIW
 - Programs are compiled for CISC ISA
 - Example: execute code machine for x86 on VLIW hardware
 - Implementation
 - Dynamic translation of CISC instructions to VLIW instructions
 - Translations cache
 - Dynamic optimization of CISC code
 - Transparent to BIOS, SO and users applications
 - Thin software translation layer – CPU firmware

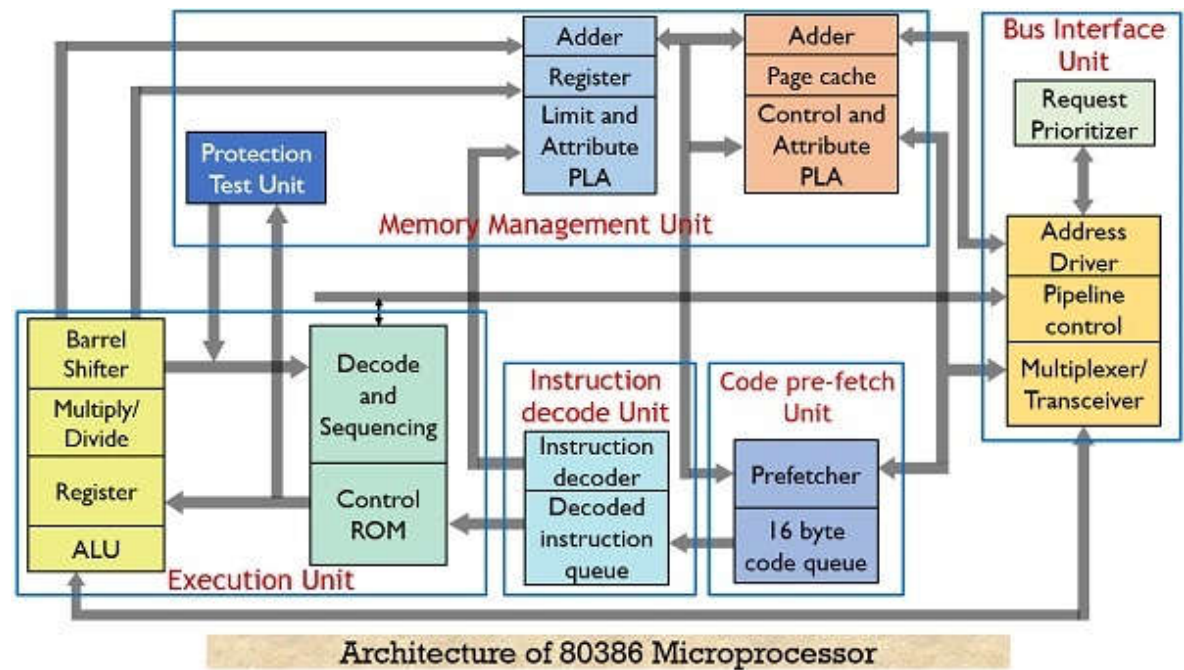
Classification

- CMS
 - Online code execution monitoring
 - Software level dynamic code optimization
 - Transmeta – Crusoe, Efficeon, LongRun



Internal architecture

- 80386
 - Bus Interface Unit
 - Code Fetch Unit
 - Instruction Decode Unit
 - Execution Unit
 - Memory Management Unit



Architecture of 80386 Microprocessor

Internal architecture

- 80386 – pipeline
 - Three stage pipeline
 - Fetch
 - Decode
 - Execution

	Cycle 1	Cycle 2	Cycle 3	Cycle 4	Cycle 5
Instruction 1	Fetch	Decode	Execute		
Instruction 2		Fetch	Decode	Execute	
Instruction 3			Fetch	Decode	Execute

Internal architecture

- Bus interface unit (BIU)
 - Whenever the CPU needs an instruction or a data it generates signals for activating the data and address bus in order to fetch the data from the desired address.
 - Generates 32 bits physical address for each bus transfer
 - Provides 32 bits bidirectional data bus
 - Generates control signals for external memories and I/O ports according to the previous discussed machine cycles (bus cycles)

Internal architecture

- Code Prefetch Unit
 - Fetches the instructions stored in the memory by making use of system buses.
 - When the CPU needs an instruction to execute, this unit fetches that instruction from the memory and stores it in 16-byte prefetch queue.
 - To speed up the operation this unit fetches the instructions in advance and the queue stores these instructions.
 - The sequence in which the instructions are fetched and gets stored in the queue depends on the order they exist in the memory.
 - Program instruction order

Internal architecture

- Instruction Decode Unit
 - Programs are executed from the main memory
 - Programs are encoded in a sequence of bytes called machine code (binary code/instruction code)
 - Decode unit takes instruction codes from prefetch queue then decodes and translate them into internal micro-operations
 - Every instruction is decoded into one or more micro-operations
 - The execution unit executes micro-operations

Internal architecture

- Memory management unit
 - Segmentation unit
 - Support for logical addresses used by programs
 - All modes (real and protected)
 - Paging unit
 - Support for virtual memory implementation
 - Protected mode

Internal architecture

- Segmentation unit
 - Has been introduced from a practical reason
 - Generate physical addresses (20 bits) using 16 bits registers
 - Split the flat address space of the processor in blocks called segments
 - It offers protection mechanism in order to protect the code or data present in the memory from application programs.

Internal architecture

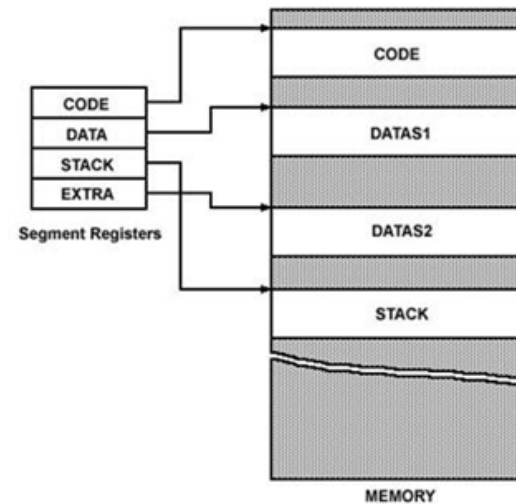
- Memory management unit – segmentation unit
 - Address computation ALU
 - Segment registers
 - Instruction pointer
 - Offset address within code segment
 - Points to the next instruction to be executed (fetched)

Internal architecture

- Segment registers
 - 16 bits registers
 - Code segment register (CS)
 - Data segment register (DS)
 - Extra data segment register (ES, FS, GS)
 - Stack segment register (SS)

Internal architecture

- Memory segmentation – 8086/real mode
 - 1 MB memory space is split in segments
 - Every segment has max. 64 KB
 - Segments can overlap each other



Internal architecture

- Memory segmentation
 - Segments hold
 - Code – program instruction code
 - Data – variables and data structures for the program
 - Stack – program stack (function calls, function parameters, local variables)

Internal architecture

- Memory segmentation
 - Physical address
 - The address output by the BIU on address bus lines
 - The address used to select the physical memory location
 - Logical address
 - The address of the memory operands specified for program instructions
 - The logical address is formed of
 - Segment
 - Offset

Internal architecture

- Memory segmentation

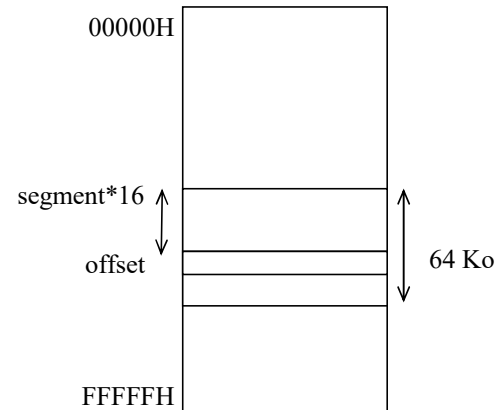
- Real mode (8086 mode)

- $\text{physical address} = \text{segment} * 16 + \text{offset}$

- $\text{physical address} = \text{segment} \ll 4 + \text{offset}$

- Physical address

- 20 bits address (8086)
 - 24 bits (286)
 - 32 bits (IA-32)
 - 40-64 (AMD-64)



Internal architecture

- Memory segmentation
 - The first logical address in a segment has the offset 0000H
 - The starting physical address of a segment is divisible by 16 (segment * 16)
 - An offset address gives the displacement from the address base of the segment to the desired location within it

Internal architecture

- Memory segmentation
 - Which is the physical address of 8000H:1234H?
 - What logical address can be used to access memory location 48020H?
 - Provide two logical addresses to access memory location 80808H

Internal architecture

- Memory segmentation
 - A logical address maps to one physical address
 - A physical address corresponds to several logical addresses
 - How many logical addresses corresponds to a specific physical address?
 - What are the advantages and disadvantages of memory segmentation?

Internal architecture

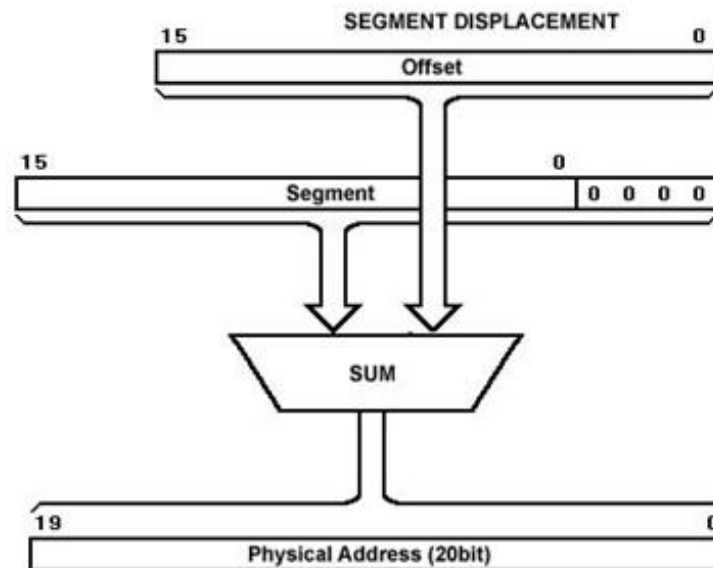
- Advantages of memory segmentation
 - Separation between code and data/ stack memory blocks
 - Programs that reference logical offset addresses only can be loaded and run anywhere in memory
 - Offset addresses always range from 00000h to max_offset, independent of the code segment base
 - Such programs are said to be relocatable, meaning that they will run at any location in memory

Internal architecture

- Disadvantages of memory segmentation
 - Limited to blocks of 64 KB (code or data) – for 16bits uP
 - Addressing larger blocks needs multiple segments manipulation
 - Physical address computation

Internal architecture

- Address computation unit



Internal architecture

- Execution Unit
 - Register set
 - ALU

Internal architecture

- Execution unit
 - Takes the next instruction from the decoder
 - Executes instructions' micro-operations
 - Stores the results in the internal registers or memory locations

Internal architecture

- Internal registers
 - Execution unit
 - General purpose registers
 - Index registers
 - Pointer registers
 - Flags register
 - Segmentation unit
 - Segment registers

Internal architecture

- Internal registers
 - 16 bits registers names: AX, BX, CX, DX



Internal architecture

- General registers roles
 - AX – accumulator (for arithmetical instruction)
 - BX – base address (for logical address computation)
 - CX – counter
 - DX – data/ multiply / I/O address

Internal architecture

- Internal registers roles
 - SI – source index
 - DI – destination index
 - SP – stack pointer
 - BP – base pointer
 - Index vs. pointer

Internal architecture

- Flags register
 - The state of the processor is stored in a dedicated register
 - Every bit of this register has specific signification
 - Status flags
 - Control flags



Internal architecture

- Flags register – status flags
 - Carry flag (CF) – set when the result of an *unsigned* arithmetic operation is too large to fit into the destination.
 - Overflow flag (OF) – set when the result of a *signed* arithmetic operation is too wide to fit into the destination.
 - Sign flag (SF) – set when the result of an arithmetic or logical operation generates a negative result.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
				OF	DF	IF	TF	SF	ZF		AF		PF		CF

Internal architecture

- Flags register – status flags
 - Zero flag (ZF) – set when the result of an arithmetic or logical operation is zero.
 - Auxiliary carry flag (AF) – set when the result of an operation causes a carry from bit 3 to bit 4.
 - Parity flag (PF) – reflects whether the number of 1 bits in the result of an operation is even or odd. 1 – odd, 0-even.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
				OF	DF	IF	TF	SF	ZF		AF		PF		CF

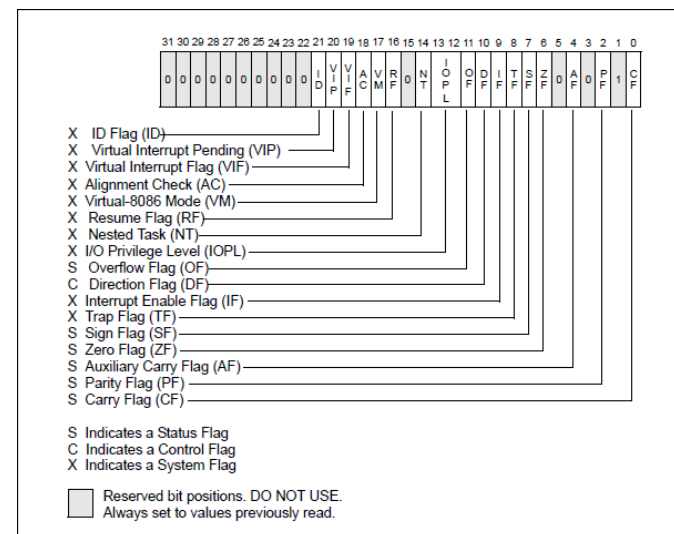
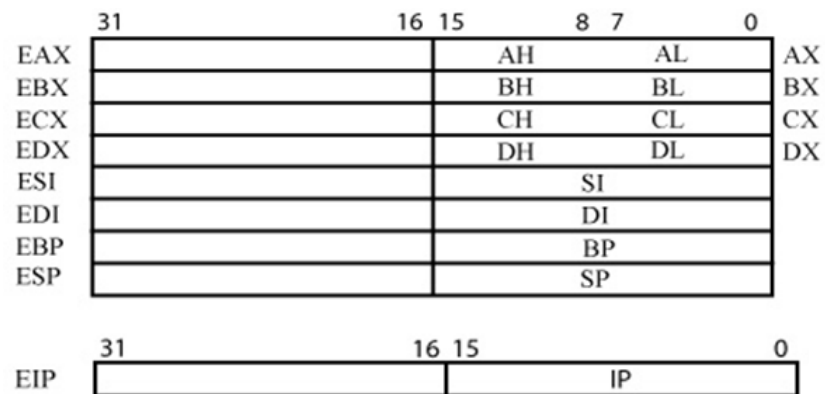
Internal architecture

- Flags register – control flags
 - **Interrupt** (IF) – dictates whether or not system interrupts can occur. 1 – enabled, 0 – disabled.
 - **Trap** (TF) – determines whether or not the CPU is halted after each instruction. Allows programmers to do tracing.
 - **Direction** (DF) – affects block data transfer instructions such as MOVS, CMPS.
 - 0 – up, 1 – down.



Internal architecture

- 32 bits registers



EFLAGS Register

Internal architecture

- Example

```
MOV EAX, 0
```

```
MOV AX, 1234H
```

```
MOV AL, 0abh
```

```
EAX = ?
```

Internal architecture

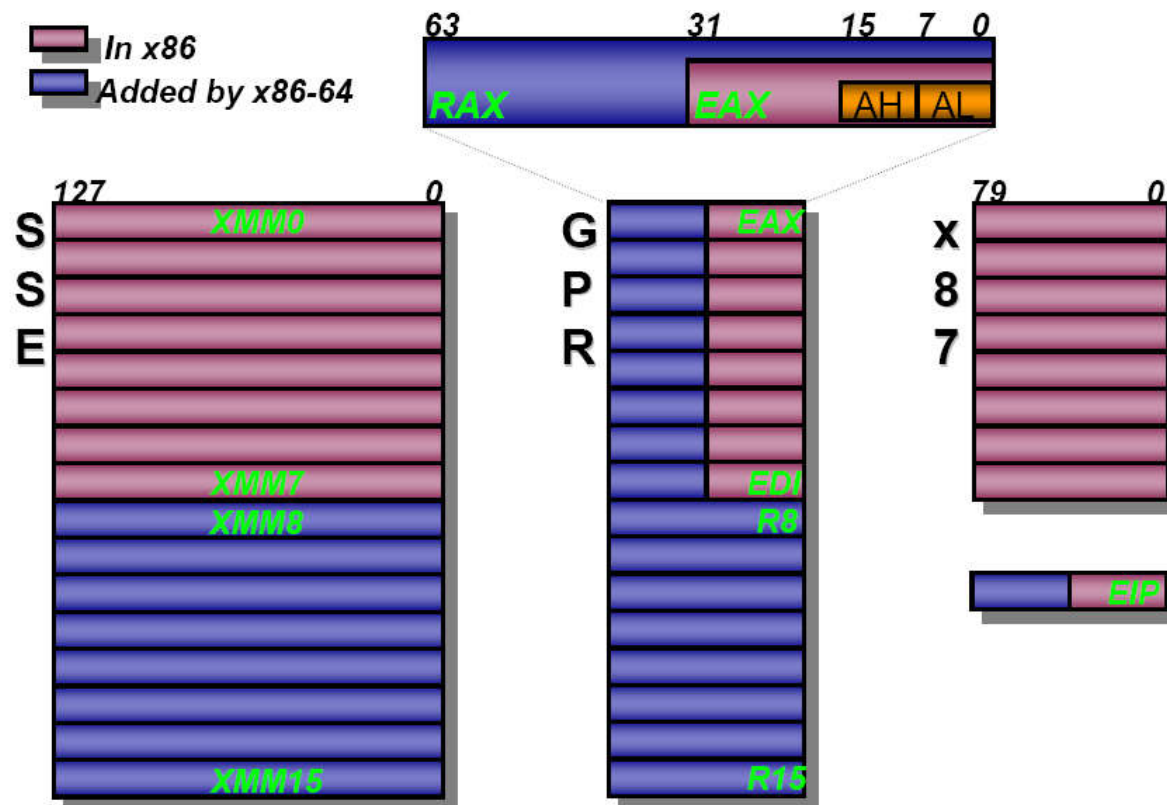
- 32 bits general registers
 - EAX – accumulator
 - EBX – base address
 - ECX – counter
 - EDX – data/ multiply / I/O address

Internal architecture

- 32 bits internal registers
 - ESI – source index
 - EDI – destination index
 - ESP – stack pointer
 - EBP – base pointer

Internal architecture

- 64 bits registers



Summary

- Definitions
 - Microprocessor
 - Bus
 - ISA
- ISA Classification
 - CISC, RISC
 - VLIW
 - EPIC-Itanium
 - CMS

