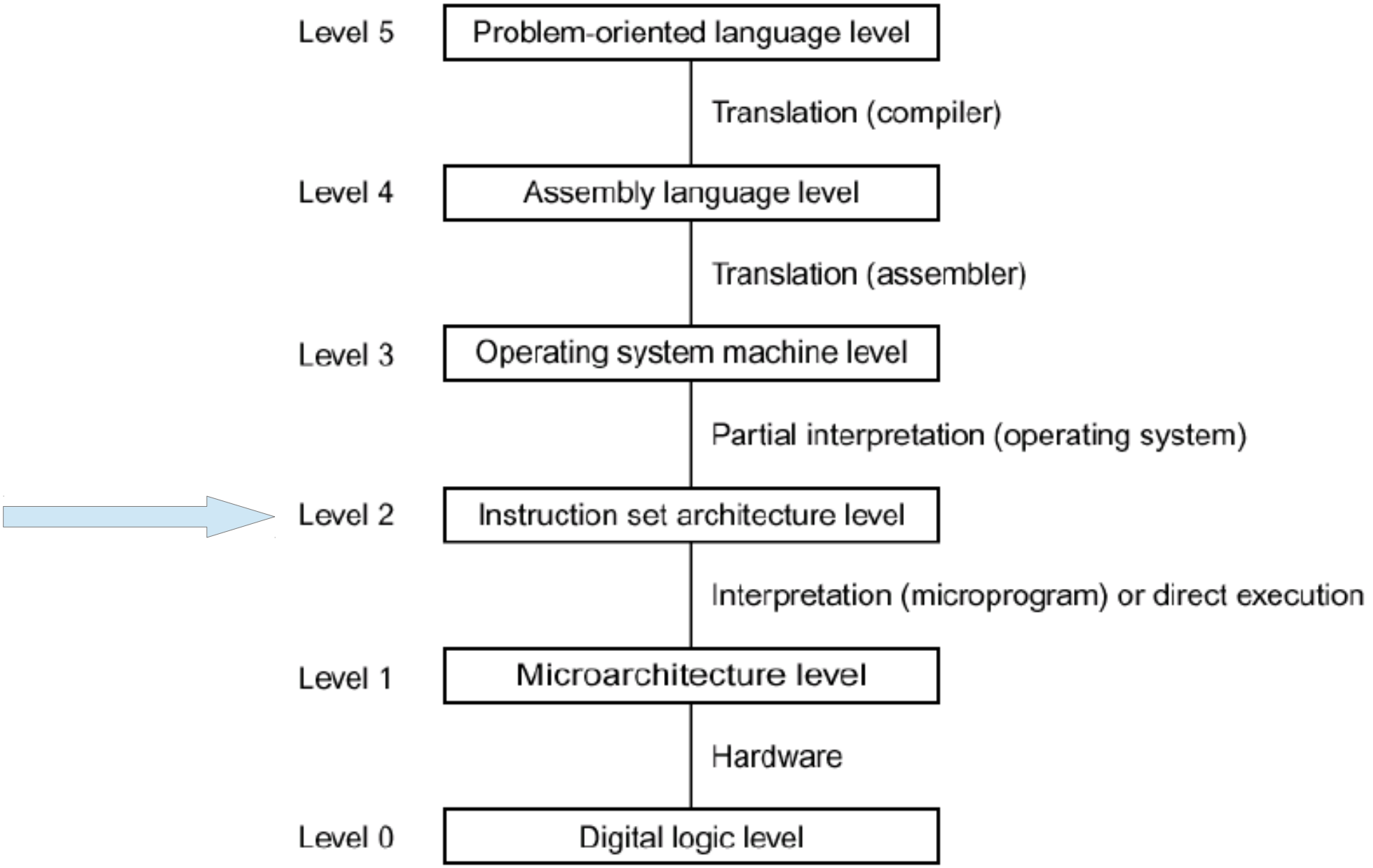
Computer Architecture

Ch4 – The Instruction Set Architecture Level

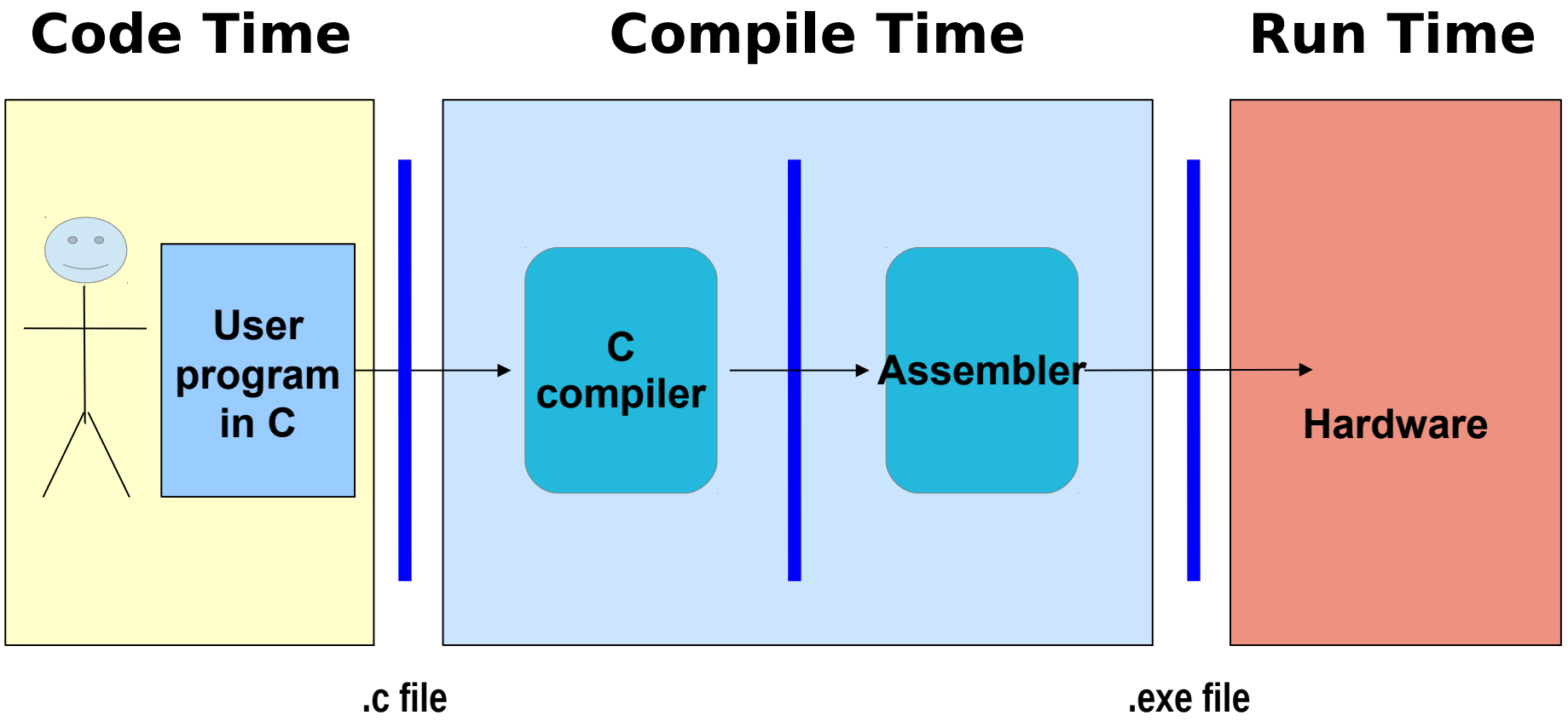
Nguyễn Quốc Đính, FIT – IUH

HCMC, Aug 2015

Where We Are


Level 5
Problem-oriented language level
Translation (compiler)
Level 4
Assembly language level
Translation (assembler)
Level 3
Operating system machine level
Partial interpretation (operating system)
Level 2
Instruction set architecture level
Interpretation (microprogram) or direct execution
Level 1
Microarchitecture level
Hardware
Level 0
Digital logic level

Translation



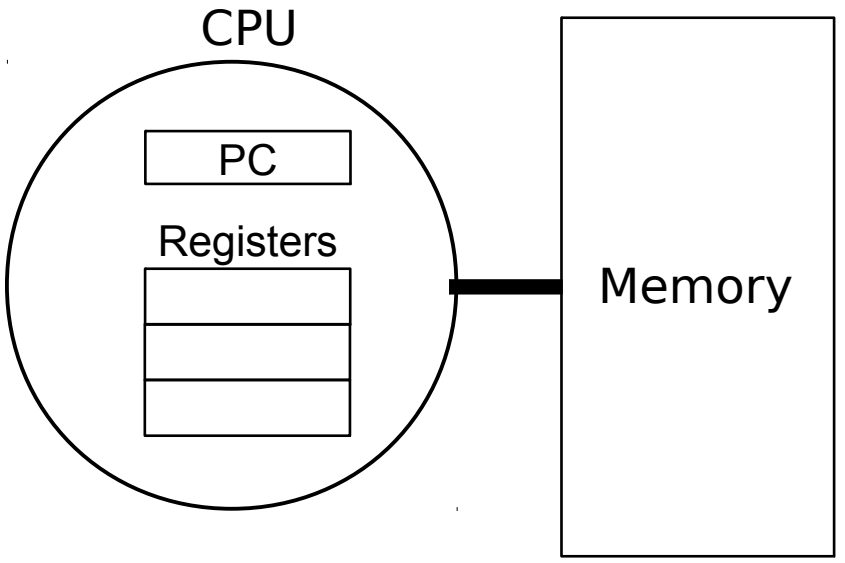
What makes programs run fast?

Translation Impact Performance

- The time required to execute a program depends on:
  - The program (as written in C, for instance)
  - The compiler: what set of assembler instructions it translates the C program into
  - The instruction set architecture (ISA): what set of instructions it makes available to the compiler
  - The hardware implementation: how much time it takes to execute an instruction

Instruction Set Architecture

- The ISA defines:
  - The system's state (e.g. registers, memory, program counter)
  - The instructions the CPU can execute
  - The effect that each of these instructions will have on the system state



General ISA Design Decisions

- **Instructions**
  - What instructions are available? What do they do?
  - How are they encoded?
- **Registers**
  - How many registers are there?
  - How wide are they?
- **Memory**
  - How do you specify a memory location?

x86

- **Processors that implement the x86 ISA completely dominate the server, desktop and laptop markets**

- Evolutionary design
  - Backwards compatible up until 8086, introduced in 1978
  - Added more features as time goes on

- **Complex instruction set computer (CISC)**
  - Many different instructions with many different formats
  - But, only small subset encountered with Linux programs
  - (as opposed to Reduced Instruction Set Computers (RISC), which use simpler instructions)

Intel x86 Evolution

Name	*Date*	*Transistors*	*MHz*
8086	**1978**	**29K**	**5-10**
386	**1985**	**275K**	**16-33**
Pentium 4F	**2005**	**230M**	**2800-3800**

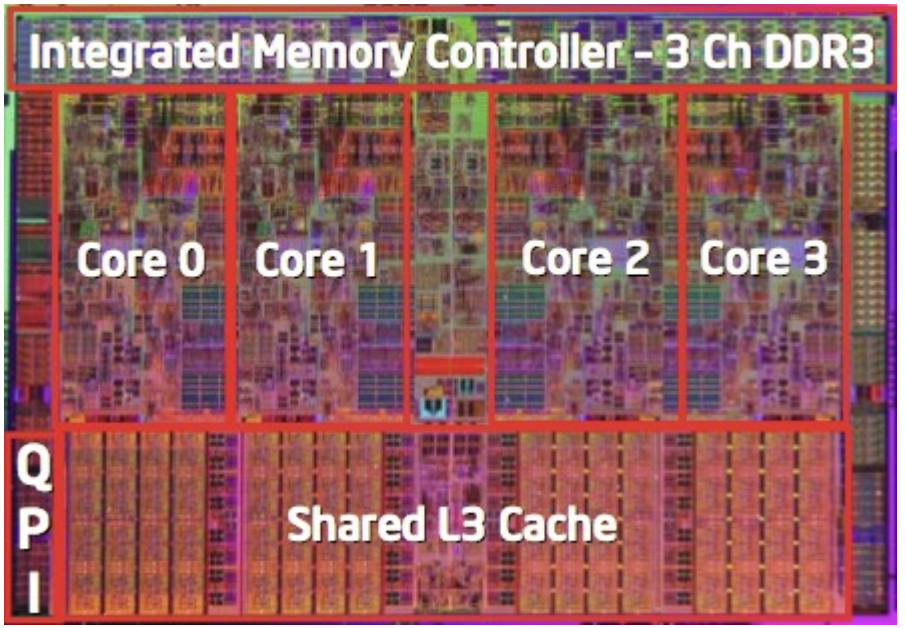
- First 16-bit processor. Basis for IBM PC & DOS
- 1MB address space
- First 32 bit processor, referred to as IA32
- Added “flat addressing”
- Capable of running Unix
- 32-bit Linux/gcc targets i386 by default
- First 64-bit Intel x86 processor, referred to as x86-64

Intel x86 Processors

- **Machine Evolution**

486	1989	1.9M
Pentium	1993	3.1M
Pentium/MMX	1997	4.5M
PentiumPro	1995	6.5M
Pentium III	1999	8.2M
Pentium 4	2001	42M
Core 2 Duo	2006	291M
Core i7	2008	731M

Intel Core i7



- **Added Features**
  - Instructions to support multimedia operations
  - Parallel operations on 1, 2, and 4-byte data
  - Instructions to enable more efficient conditional operations
  - More cores!

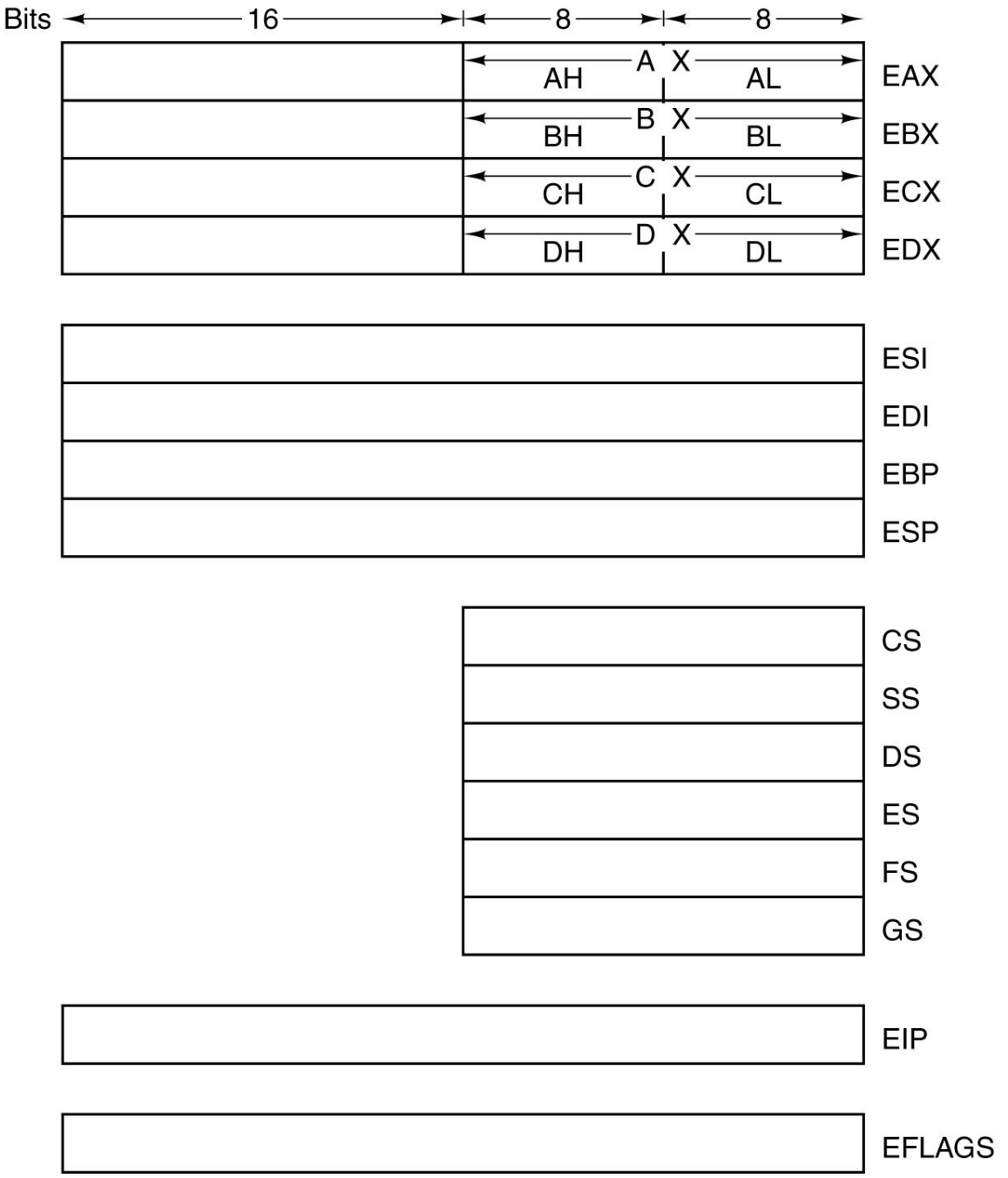
x86 Clone: Advanced Micro Devices (AMD)

- Historically
  - AMD has followed just behind Intel
  - A little bit slower, a lot cheaper
- Then
  - Recruited top circuit designers from Digital Equipment and other downward trending companies
  - Built Opteron: tough competitor to Pentium 4
  - Developed x86-64, their own extension of x86 to 64 bits

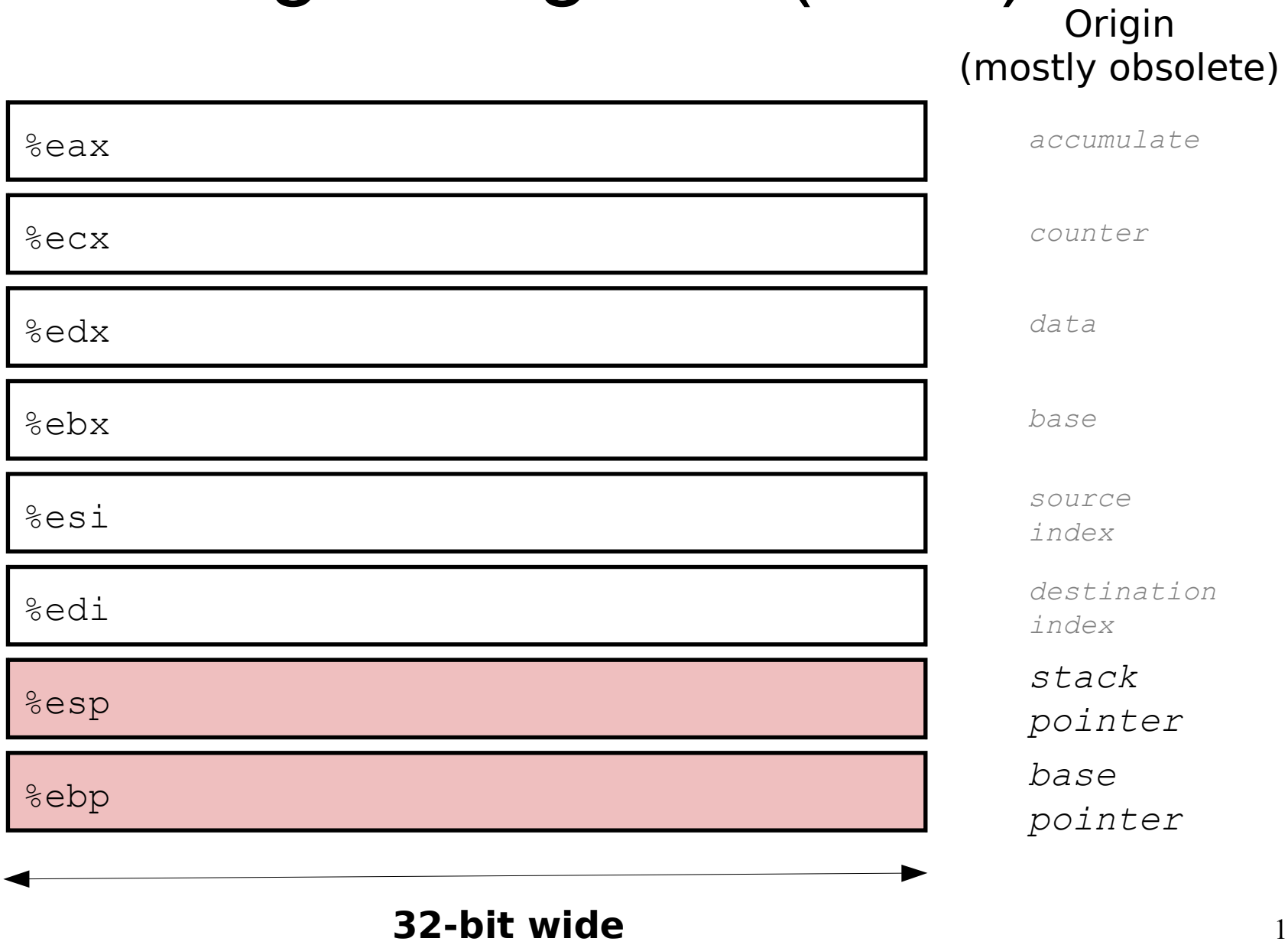
- Issues in new ISA Design and Application
  - “Is it compatible with it’s own predecessor?”
  - “Can I run my old OS on it?”
  - “Will it run existing applications unmodified?”
- Backward Compatibility
  - Source code compatible
  - Machine code compatible
- A new ISA will have new features, but what about other aspects of the language!
  - What remains and what changes?
  - A revolutionary change may make all higher level software change!

Overview of the Pentium 4 ISA Level

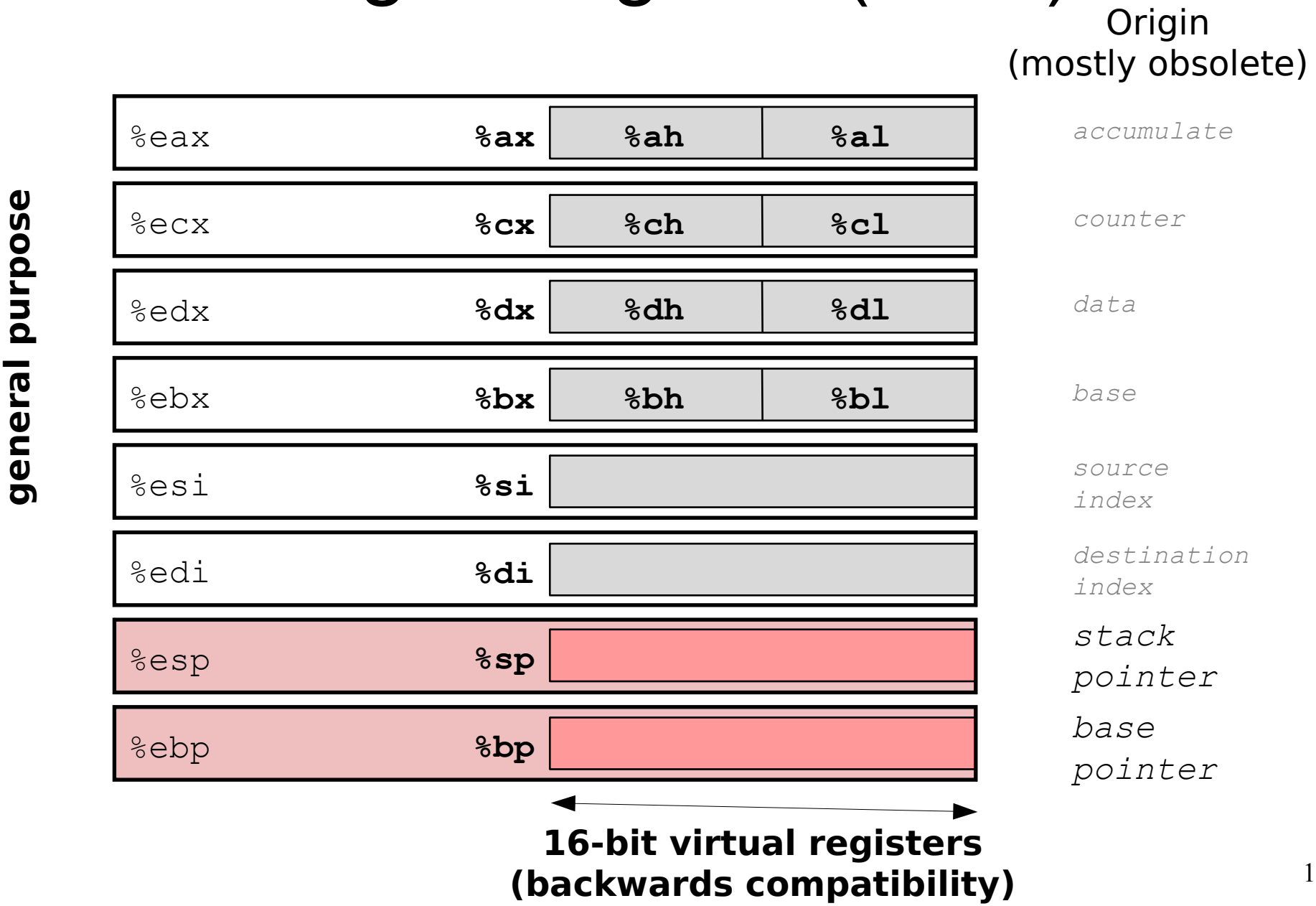
The Pentium 4's primary registers.



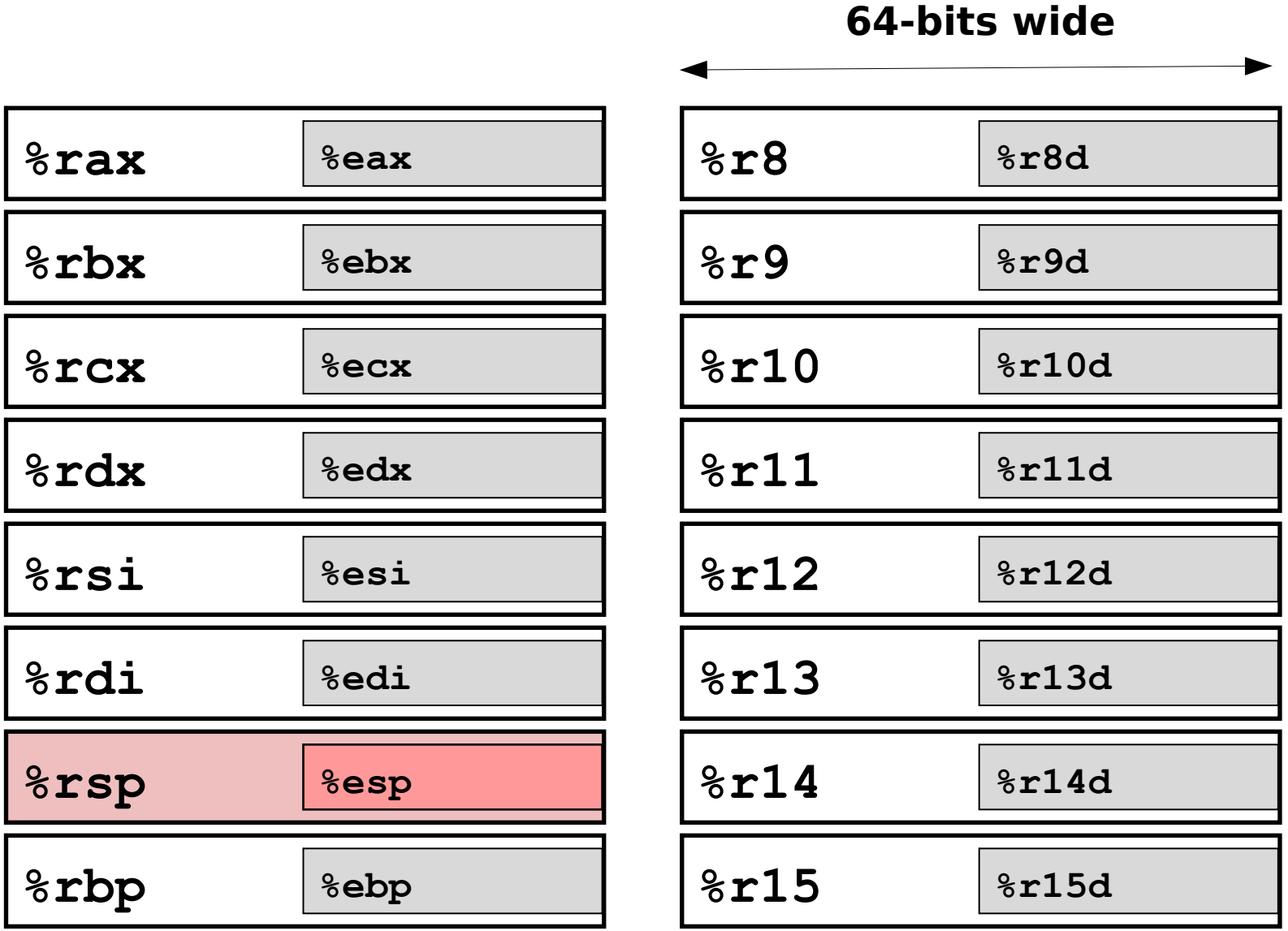
Integer Register (IA32)



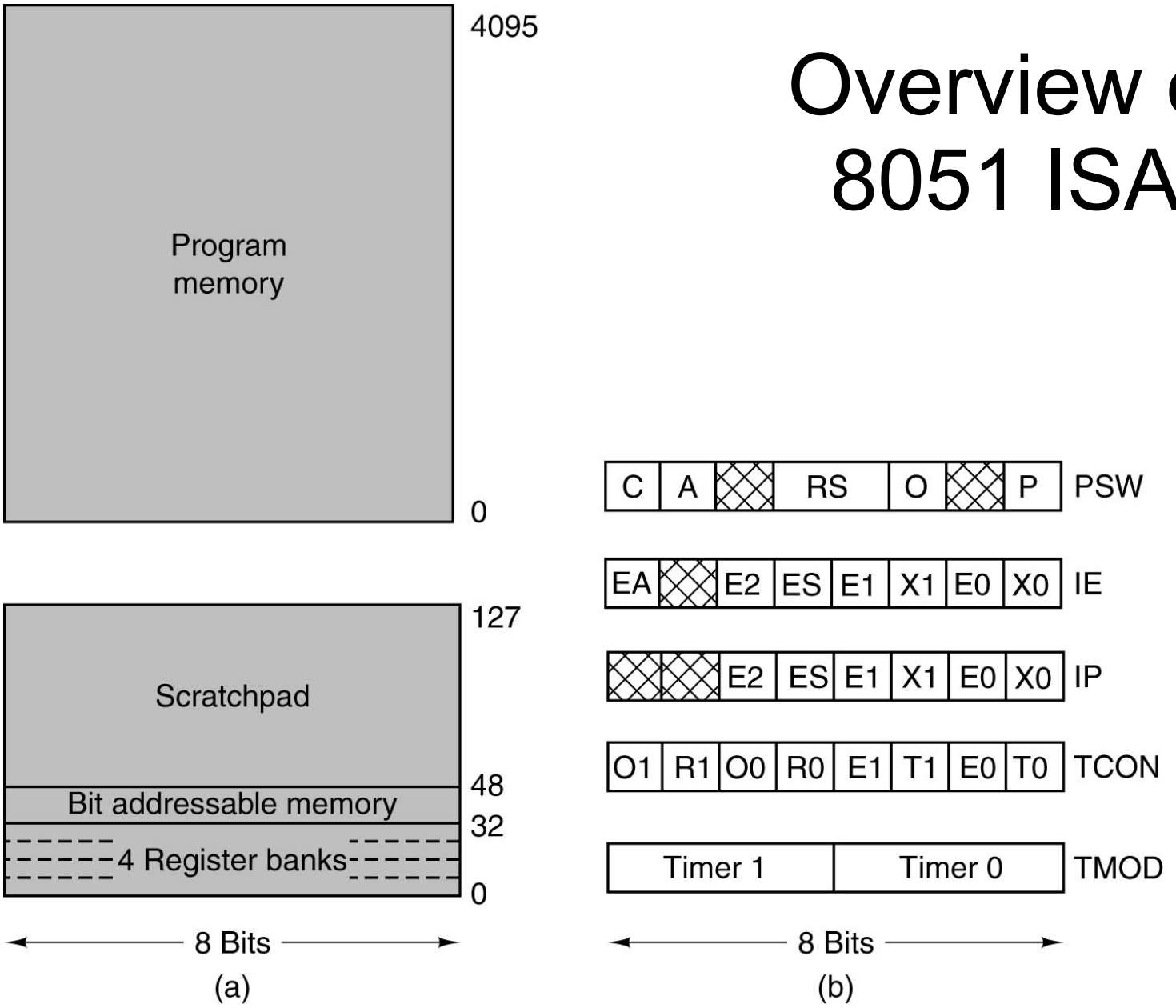
Integer Register (IA32)



x86-64 Integer Register



Overview of the 8051 ISA Level



(a) On-chip memory organization for the 8051.
(b) Major 8051 registers.

Data Types on the Pentium 4

Type	1 Bit	8 Bits	16 Bits	32 Bits	64 Bits	128 Bits
Bit						
Signed integer		×	×	×		
Unsigned integer		×	×	×		
Binary coded decimal integer		×				
Floating point				×	×	

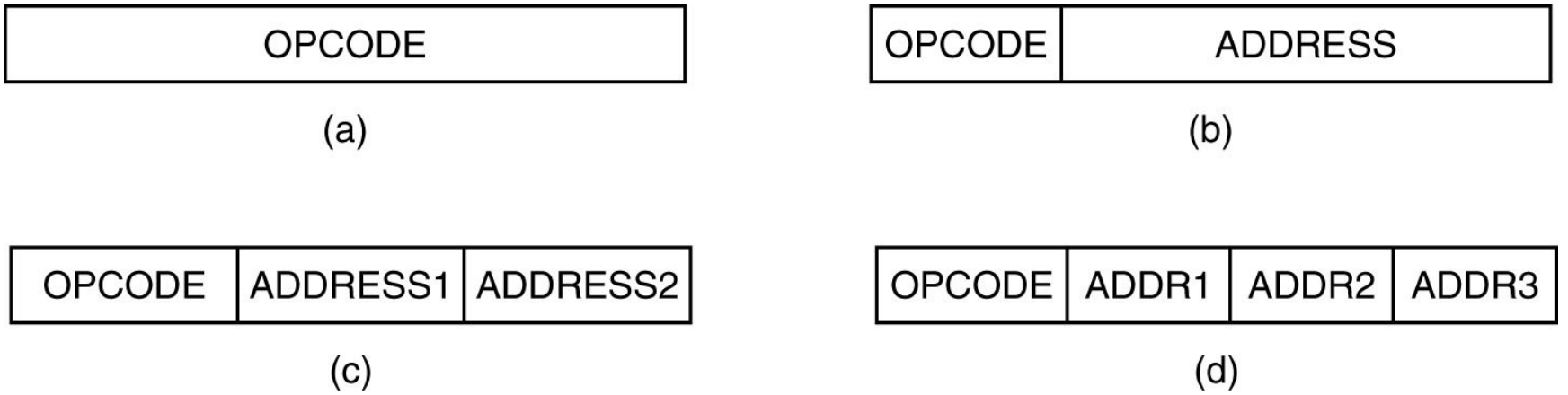
The Pentium 4 numeric data types.
Supported types are marked with ×.

Data Types on the 8051

Type	1 Bit	8 Bits	16 Bits	32 Bits	64 Bits	128 Bits
Bit	×					
Signed integer		×				
Unsigned integer						
Binary coded decimal integer						
Floating point						

The 8051 numeric data types.
Supported types are marked with ×.

Instruction Format (1)



Four common instruction formats:
(a) Zero-address instruction. (b) One-address instruction
(c) Two-address instruction. (d) Three-address instruction.

Instruction Addressing

4-Address:

- Conventional ALU
- two addresses for the source
- one address for the destination
- one address for the next instruction

3-Address:

- Conventional ALU
- two addresses for the source
- one address for the destination
- implied next instruction (or an alternate instruction format)

2-Address:

- Conventional ALU
- two addresses for the source
- results overwrite one of the sources

1-Address:

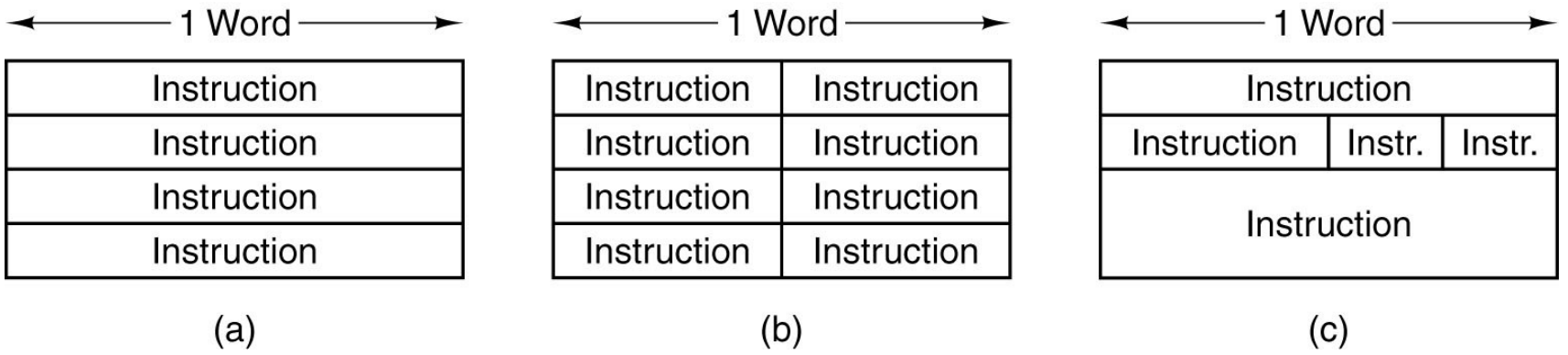
- Accumulator Based machines
- one address is used for one source
- the accumulator provides the other and the destination

0-Address:

- Stack Based machines
- use the stack as both source and destination

Instruction Format (2)

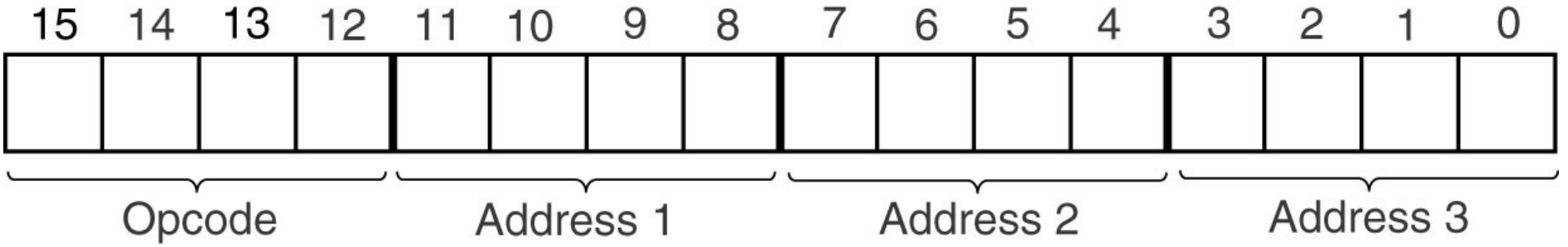
- **Simple RISC processors:**
  - All instructions the same size
  - The typical instruction is only one word
- **CISC processors:**
  - Variable word size instructions
  - Typically 1, 2, 3, or 4 words
- **Unique Languages:**
  - Multiple Instructions per word
  - ARM core 32-bit machine has some 16-bit instructions



Some possible relationships between instruction and word length.

Expanding Opcodes (1)

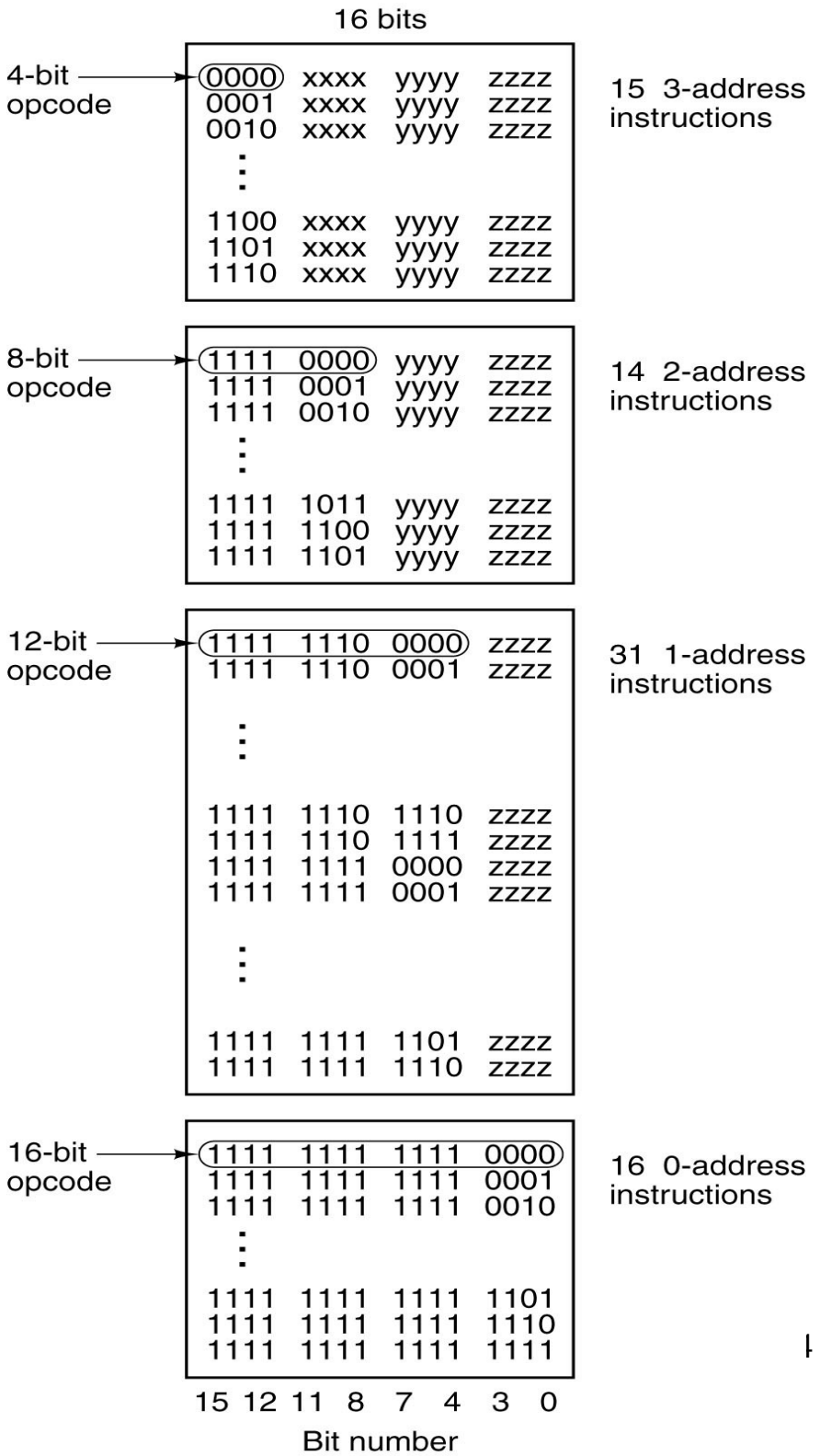
- Assume:
  - 16-bit instruction
  - 16 registers (4-bits to ID)
  - a mix of 3-, 2-, 1-, and 0-address instructions



An instruction with a 4-bit opcode and three 4-bit address fields.

Expanding Opcodes (2)

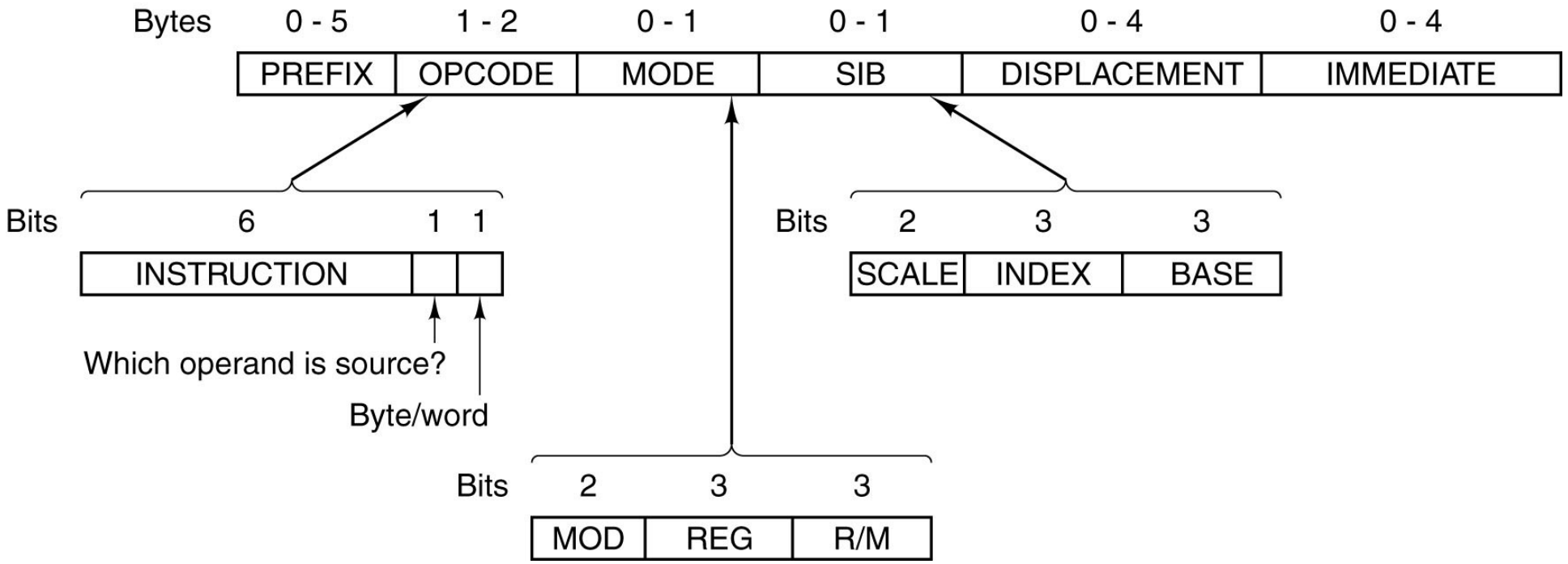
An expanding opcode allowing 15 three-address instructions, 14 two-address instructions, 31 one-address instructions, and 16 zero-address instructions. The fields marked *xxxx*, *yyyy*, and *zzzz* are 4-bit address fields.



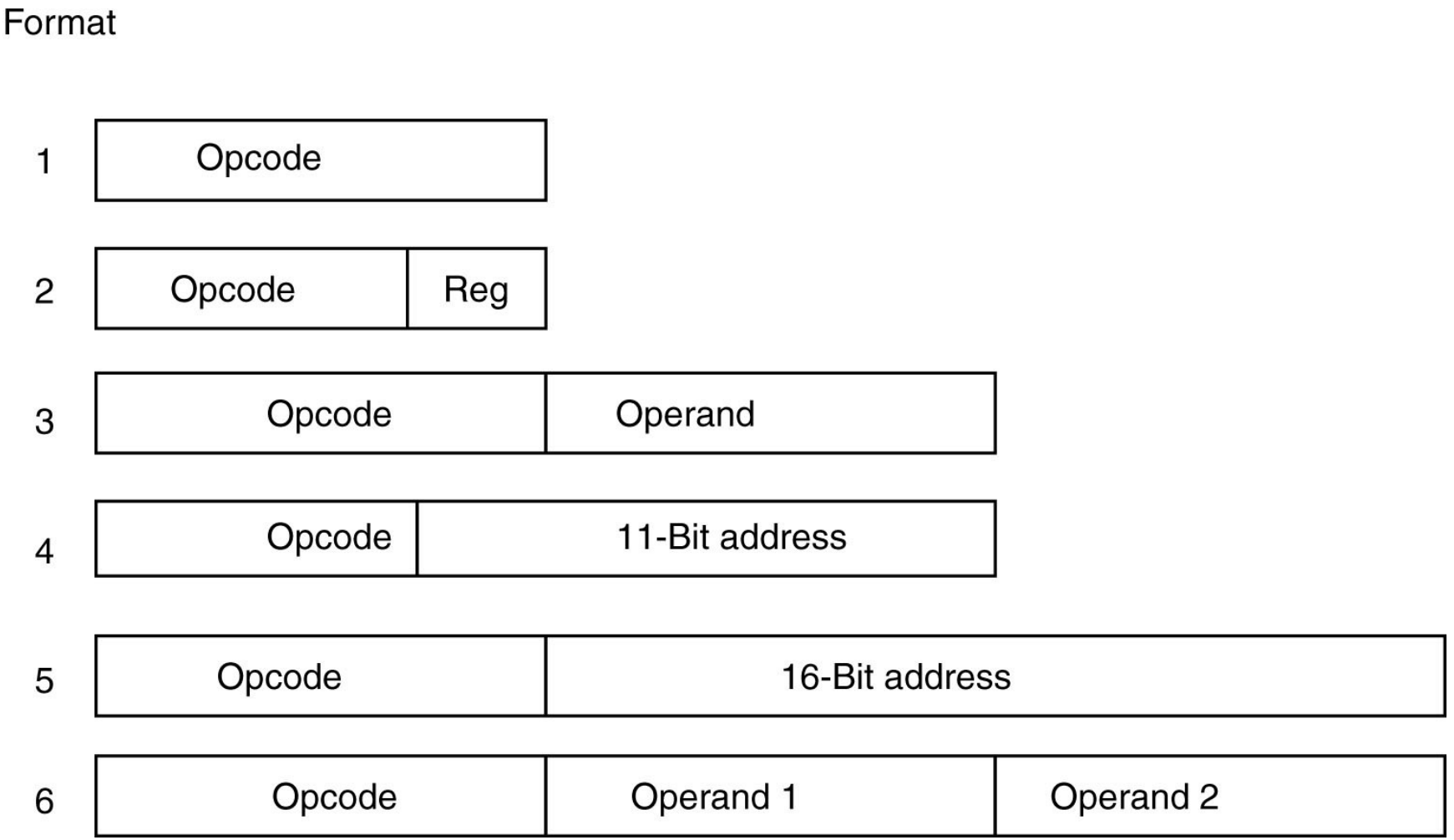
Expanding Opcodes (3)

- Result:
  - There are four instruction formats
  - One format for 3-, 2-, 1-, and 0- address instructions each
  - Formats 1, 2, and 3 have 15 instructions available
  - Format 0 has 16 instructions available
- Total instructions
  - 15 3-address instructions
  - 15 2-address instructions
  - 15 1-address instructions
  - 16 0-address instructions
  - Total of 61 instructions available

Pentium 4 Instruction Formats



8051 Instruction Format



Addressing

The most significant part of the instruction!

Example (Heuring and Jordan)

Immediate Addressing

Value ← #XXXX

Direct Addressing

Value ← M[#XXXX]

Register Addressing

Value ← R(a)

Register Indirect Addressing

Value ← M[R(a)]

Indexed Addressing

Value ← M[R(a) + #XXXX]

Based-Indexed Addressing

Value ← M[R(a) + R(b)]

PC Relative Addressing

NewPC ← PC + #XXXX

A generic assembly program for computing the R1 OR of (*Ai* AND *Bi)* for two 1024-element arrays.

```
        MOV R1,#0          ; accumulate the OR in R1, initially 0
        MOV R2,#0          ; R2 = index, i, of current product: A[i] AND B[i]
        MOV R3,#4096       ; R3 = first index value not to use
LOOP:   MOV R4,A(R2)       ; R4 = A[i]
        AND R4,B(R2)       ; R4 = A[i] AND B[i]
        OR R1,R4           ; OR all the Boolean products into R1
        ADD R2,#4          ; i = i + 4 (step in units of 1 word = 4 bytes)
        CMP R2,R3          ; are we done yet?
        BLT LOOP           ; if R2 < R3, we are not done, so continue
```

Register, Immediate
Register, Immediate
Register, Immediate
LOOP: Register, Indexed (Base and displacement)
Register, Indexed (Base and displacement)
Register, Register
Register, Immediate
Register, Register
PC Relative

A generic assembly program for computing the sum of the elements of an array.

```
        MOV R1,#0          ; accumulate the sum in R1, initially 0
        MOV R2,#A          ; R2 = address of the array A
        MOV R3,#A+4096     ; R3 = address of the first word beyond A
LOOP:   ADD R1,(R2)        ; register indirect through R2 to get operand
        ADD R2,#4          ; increment R2 by one word (4 bytes)
        CMP R2,R3          ; are we done yet?
        BLT LOOP           ; if R2 < R3, we are not done, so continue
```

Q: Find addressing model identification on each line

Discussion of Addressing Models

Addressing mode	Pentium 4	UltraSPARC III	8051
Accumulator			×
Immediate	×	×	×
Direct	×		×
Register	×	×	×
Register indirect	×	×	×
Indexed	×	×	
Based-indexed		×	
Stack			

A comparison of addressing modes.

Loop Control

```
        i = 1;
L1:     first-statement;
        .
        .
        .
        last-statement;
        i = i + 1;
        if (i < n) goto L1;
```

(a)

```
        i = 1;
L1:     if (i > n) goto L2;
        first-statement;
        .
        .
        .
        last-statement
        i = i + 1;
        goto L1;
L2:
```

(b)

(a) Test-at-the-end loop.

The loop operations will be executed at least once ... is it desirable?

(b) Test-at-the-beginning loop.

The loop instructions will not be executed if the conditional does not pass.

Input/Output Instruction

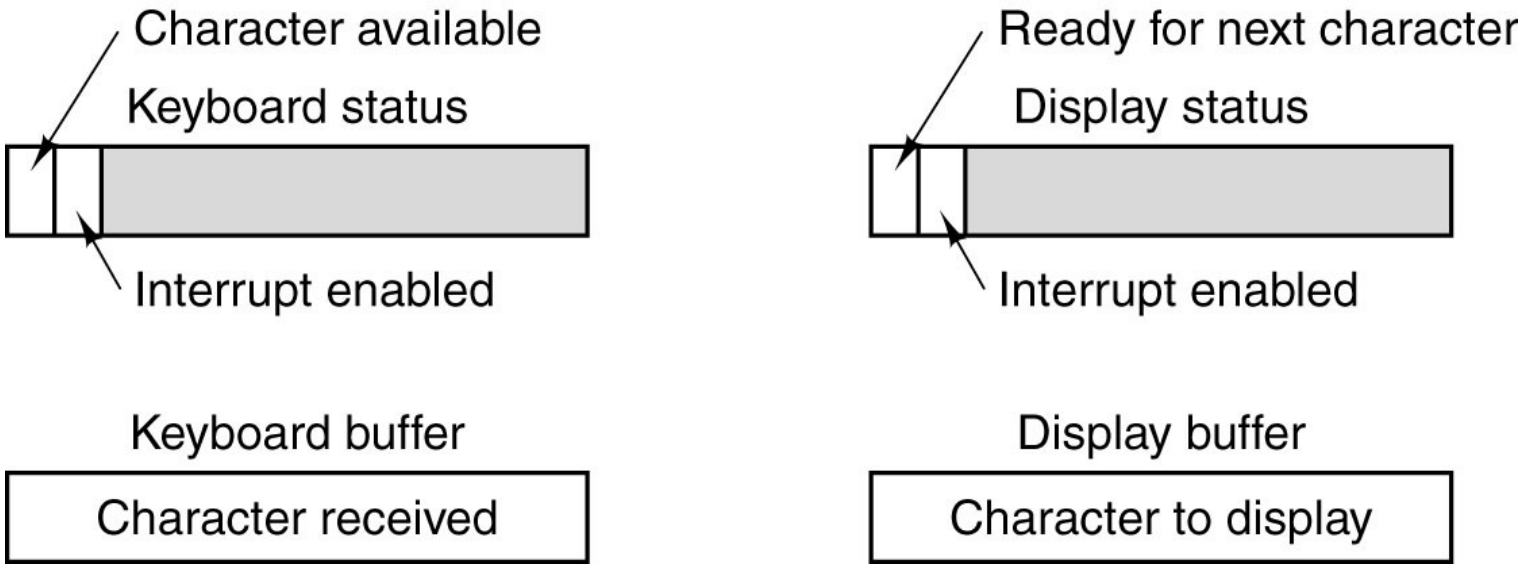
There are three distinct classes of I/O operations:

1. Programmed I/O with busy waiting
2. Interrupt-driven I/O
3. DMA I/O

Programmed I/O

- Commonly used in low-end microprocessors, for example, in embedded systems or in systems that must respond quickly to external changes (real-time)
- Data and status are “read” or “written” from defined I/O address locations (typically Memory Mapped I/O addresses).
- The normal procedure requires:
  - Check a bit in a status register
  - If the bit check does not pass defined criteria, keep checking (a loop)
  - Once the status passes, read or write the data

Programmed I/O



Example routines (keyboard, mouse, modem, UART, etc.):

get_character: read a word or character from an I/O device

put_character: write a word or character from an I/O device

Programmed I/O

```
public static void output_buffer(char buf[ ], int count) {
    // Output a block of data to the device
    int status, i, ready;
    for (i = 0; i < count; i++) {
        do {
            status = in(display_status_reg);
            // get status
            ready = (status >> 7) & 0x01;
            // isolate ready bit
        } while (ready != 1);
        out(display_buffer_reg, buf[i]);
    }
}
```

An example of programmed I/O.

Programmed I/O

- Disadvantages:
  - CPU spends most of its time in a tight loop waiting for the device to become ready, call **busy waiting**
  - If the CPU has nothing else to do (e.g., the CPU in a washing machine), busy waiting may be OK
  - if there is other work to do, such as running other programs, busy waiting is wasteful

Interrupt-driven I/O

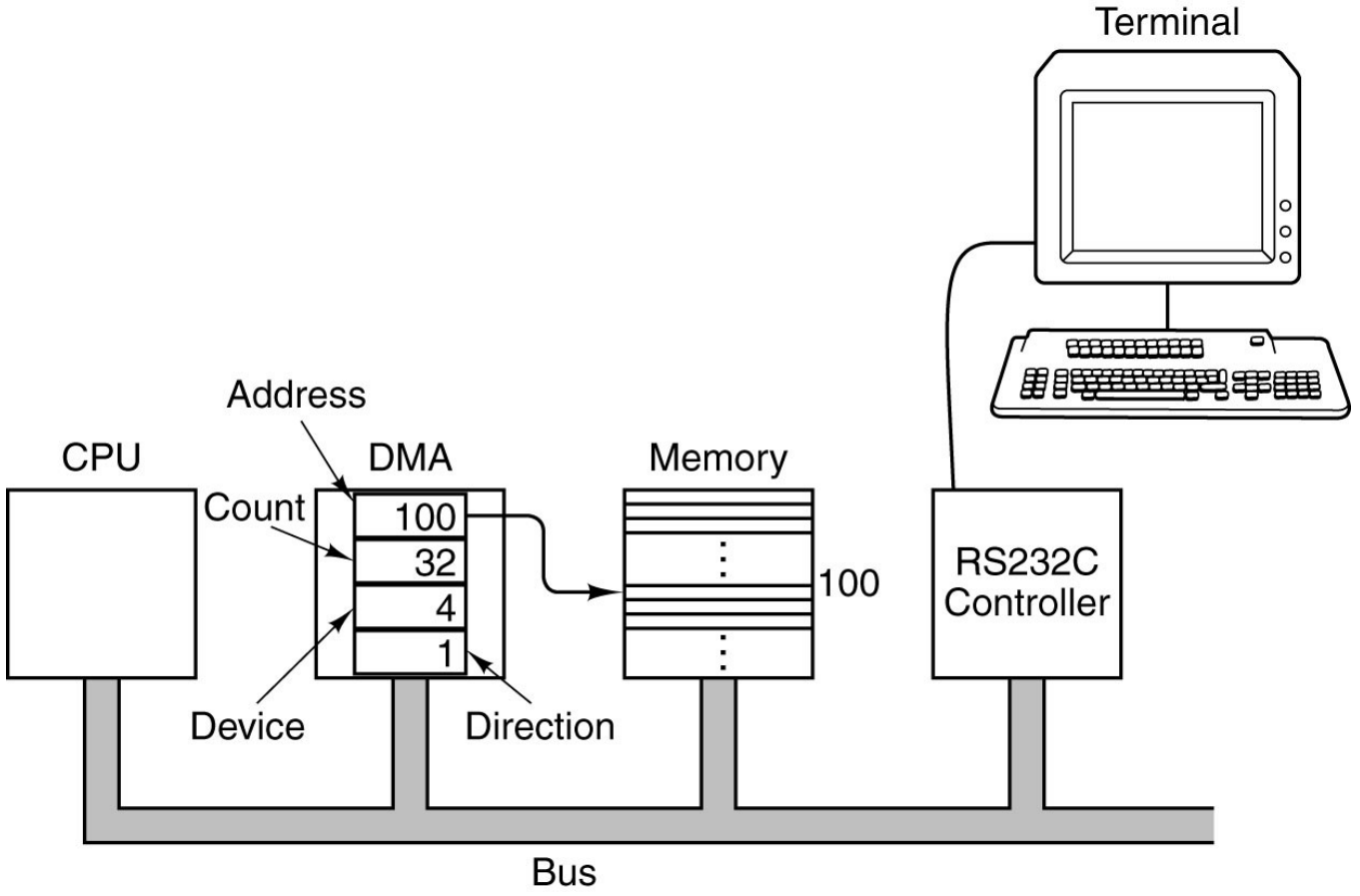
- Instead of waiting for valid status, most processors provide interrupts based on I/O status changes.

- Requirements to operate:
  - Interrupt capability
  - Initialization procedures to initialize the port and interrupt registers
  - An Interrupt Service Routine (ISR) for defined/used interrupts
  - Interrupt enabling once the processor needs to recognize/execute interrupts

Interrupt-driven I/O

- Advantages:
  - Interrupts allow the program to execute will waiting for the I/O device or status
  - Wasted clock cycles are recovered, if something useful can be accomplished
- Disadvantages:
  - Interrupts “happen when they happen”
  - The processors system must perform “Precise Interrupts” or the program will likely crash
  - Interrupts require a large number of clock cycles (wasted overhead) to enter the ISR and return from the ISR

DMA (Direct Memory Access) I/O



A system with a DMA controller. DMA is to write a block of 32 bytes from memory address 100 to a terminal (say, device 4)

DMA Chip

- The DMA chip has (at least) four registers inside it, all of which can be loaded by software running on the CPU.
  - The first one contains the memory address to be read or written.
  - The second one contains the count of how many bytes (or words) are to be transferred.
  - The third one specifies the device number or I/O space address to use, thus specifying which I/O device is desired.
  - The fourth one tells whether data are to be read from or written to the I/O device.

DMA I/O

Requirements to operate:

- DMA processing ability – shared bus with bus request and grants
- Initialization procedures to initialize the port, interrupt and DMA registers.
- (DMA input address, output address, transfer count)
- An Interrupt Service Routine (ISR) for DMA interrupts (when done!)
- DMA and interrupt enabling once the processor needs to recognize/execute interrupts

DMA I/O

- Advantages:
  - DMA allow the program to execute will waiting for the DMA I/O to complete
  - Wasted CPU and multiple interrupt clock cycles are recovered.
  - A software interrupt or non-wait polling may be used
- Disadvantages:
  - DMAs use peripherals, the data bus and memory to move data; therefore,
  - The CPU must share the bus with the DMA processor
  - Bus priority must be defined for CPU and DMA
  - (Note: if the DMA is time-critical, it must have priority over CPU bus cycles!)

The Pentium 4 Instructions (1)

Moves

MOV DST,SRC	Move SRC to DST
PUSH SRC	Push SRC onto the stack
POP DST	Pop a word from the stack to DST
XCHG DS1,DS2	Exchange DS1 and DS2
LEA DST,SRC	Load effective addr of SRC into DST
CMOVcc DST,SRC	Conditional move

Arithmetic

ADD DST,SRC	Add SRC to DST
SUB DST,SRC	Subtract SRC from DST
MUL SRC	Multiply EAX by SRC (unsigned)
IMUL SRC	Multiply EAX by SRC (signed)
DIV SRC	Divide EDX:EAX by SRC (unsigned)
IDIV SRC	Divide EDX:EAX by SRC (signed)
ADC DST,SRC	Add SRC to DST, then add carry bit
SBB DST,SRC	Subtract SRC & carry from DST
INC DST	Add 1 to DST
DEC DST	Subtract 1 from DST
NEG DST	Negate DST (subtract it from 0)

A selection of the Pentium 4 integer instructions.

The Pentium 4 Instructions (2)

Binary coded decimal

DAA	Decimal adjust
DAS	Decimal adjust for subtraction
AAA	ASCII adjust for addition
AAS	ASCII adjust for subtraction
AAM	ASCII adjust for multiplication
AAD	ASCII adjust for division

Boolean

AND DST,SRC	Boolean AND SRC into DST
OR DST,SRC	Boolean OR SRC into DST
XOR DST,SRC	Boolean Exclusive OR SRC to DST
NOT DST	Replace DST with 1's complement

Shift/rotate

SAL/SAR DST,#	Shift DST left/right # bits
SHL/SHR DST,#	Logical shift DST left/right # bits
ROL/ROR DST,#	Rotate DST left/right # bits
RCL/RCR DST,#	Rotate DST through carry # bits

A selection of the Pentium 4 integer instructions.

The Pentium 4 Instructions (3)

Test/compare

TEST SRC1,SRC2	Boolean AND operands, set flags
CMP SRC1,SRC2	Set flags based on SRC1 - SRC2

Transfer of control

JMP ADDR	Jump to ADDR
Jxx ADDR	Conditional jumps based on flags
CALL ADDR	Call procedure at ADDR
RET	Return from procedure
IRET	Return from interrupt
LOOPxx	Loop until condition met
INT n	Initiate a software interrupt
INTO	Interrupt if overflow bit is set

Strings

LODS	Load string
STOS	Store string
MOVS	Move string
CMPS	Compare two strings
SCAS	Scan Strings

A selection of the Pentium 4 integer instructions.

The Pentium 4 Instructions (4)

Condition codes

STC	Set carry bit in EFLAGS register
CLC	Clear carry bit in EFLAGS register
CMC	Complement carry bit in EFLAGS
STD	Set direction bit in EFLAGS register
CLD	Clear direction bit in EFLAGS reg
STI	Set interrupt bit in EFLAGS register
CLI	Clear interrupt bit in EFLAGS reg
PUSHFD	Push EFLAGS register onto stack
POPFD	Pop EFLAGS register from stack
LAHF	Load AH from EFLAGS register
SAHF	Store AH in EFLAGS register

Miscellaneous

SWAP DST	Change endianness of DST
CWQ	Extend EAX to EDX:EAX for division
CWDE	Extend 16-bit number in AX to EAX
ENTER SIZE,LV	Create stack frame with SIZE bytes
LEAVE	Undo stack frame built by ENTER
NOP	No operation
HLT	Halt
IN AL,PORT	Input a byte from PORT to AL
OUT PORT,AL	Output a byte from AL to PORT
WAIT	Wait for an interrupt

SRC = source
DST = destination
# = shift/rotate count
LV = # locals

A selection of the Pentium 4 integer instructions.

8051 Instructions (1)

Inst.	Description	ACC	Reg	Dir	@R	#	C	Bit
MOV	Move *src* to ACC		×	×	×	×		
MOV	Move *src* to register	×		×		×		
MOV	Move *src* to memory	×	×	×	×	×		
MOV	Move *src* to indirect RAM	×		×		×		
MOV	Move 16-bit constant to DPTR							
MOVC	Move code to ACC offset from DPTR							
MOVC	Move code to ACC offset from PC							
MOVX	Move external RAM byte to ACC				×			
MOVX	Move ext. RAM byte to ACC @DPTR							
MOVX	Move to ext. RAM byte from ACC				×			
MOVX	Move to ext. RAM byte from ACC @DPTR							
PUSH	Push *src* byte to stack			×				
POP	Pop stack byte to dst			×				
XCH	Exchange ACC and dst	×		×	×			
XCHD	Exchange low-order digit ACC and dst			×				
SWAP	Swap nibbles of dst	×						
ADD	Add *src* to ACC		×	×	×	×		

The 8051 Instruction set.

8051 Instructions (2)

Inst.	Description	ACC	Reg	Dir	@R	#	C	Bit
ADDC	Add *src* to ACC with carry		×	×	×	×		
SUBB	Subtract *src* from ACC with borrow		×	×	×	×		
INC	Increment dst	×	×	×	×			
DEC	Decrement dst	×	×	×	×			
INC	DPTR							
MUL	Multiply							
DIV	Divide							
DA	Decimal adjust dst	×						
ANL	AND *src* to ACC		×	×	×	×		
ANL	AND ACC to dst			×				
ANL	AND immediate to dst			×				
ORL	OR *src* to ACC		×	×	×	×		
ORL	OR ACC to dst			×				
ORL	OR immediate to dst			×				

The 8051 Instruction set.

8051 Instructions (3)

Inst.	Description	ACC	Reg	Dir	@R	#	C	Bit
XRL	XOR *src* to ACC		×	×	×	×		
XRL	XOR ACC to dst			×				
XRL	XOR immediate to dst			×				
CLR	Clear dst	×						
CPL	Complement dst	×						
RL	Rotate dst left	×						
RLC	Rotate dst left through carry	×						
RR	Rotate dst right	×						
RRC	Rotate dst right through carry	×						

The 8051 Instruction set.

8051 Instructions (4)

Inst.	Description	ACC	Reg	Dir	@R	#	C	Bit
CLR	Clear bit						×	×
SETB	Set bit						×	×
CPL	Complement bit						×	×
ANL	AND *src* to carry							×
ANL	AND complement of *src* to carry							×
ORL	OR *src* to carry							×
ORL	OR complement of *src* to carry							×
MOV	Move *src* to carry							×
MOV	Move carry to src							×
JV	Jump relative if carry set							
JNC	Jump relative if carry not set							
JB	Jump relative if direct bit set							×
JNB	Jump relative if direct bit not set							×
JBC	Jump rel. if direct bit set and carry clear							×

The 8051 Instruction set.

8051 Instructions (5)

Inst.	Description	ACC	Reg	Dir	@R	#	C	Bit
ACALL	Call subroutine (11-bit addr)							
LCALL	Call subroutine (16-bit addr)							
RET	Return from subroutine							
RETI	Return from interrupt							
SJMP	Short relative jump (8-bit addr)							
AJMP	Absolute jump (11-bit addr)							
LJMP	Absolute jump (16-bit addr)							
JMP	Jump indirect rel. to DPR+ACC							
JZ	Jump if ACC is zero							
JNZ	Jump if ACC is nonzero							
CJNE	Comp. *src* to ACC, jump unequal			×		×		
CJNE	Comp. *src* to immediate, jump unequal		×		×			
DJNZ	Decrement dst and jump nonzero							
NOP	No operation							

The 8051 Instruction set.

Copyright note

- Slides are adopted form lecture notes of Tanenbaum, Structured Computer Organization, Fifth Edition, (c) 2006 Pearson Education, Inc.