Advanced microprocessors

Chapter 4

Advanced Microprocessor

- Intel X86 family of advanced Microprocessor
- Programming model for 86 family.
- X85 addressing modes, instruction set, hardware.
- Motorola 68 XXX family of microprocessor
- 68 XXX addressing modes
- Instruction set and hardware.

80x86 Processor Architecture

8085 (review) – typical, single segment

8086/88 – pipeline + segments

80286/386 – real(8086)/protected mode

80386 – MMU (+paging)

80486 – cache memory

Pentium

P6 (Pentium Pro, II, Celeron, III, Xeon, ...)

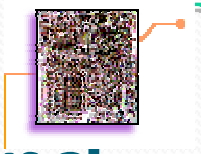
Pentium 4, Core 2 – 64 bit extension

8086 and 8088 Processors

Processor Model: BIU+EU

Programming Model:
Data Registers + Segments

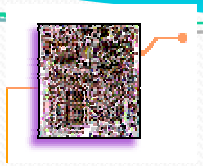
Memory Banks Issues



8086: IA standard

- Became available in 1978
  - 16-bit registers (8-/16-bit operations) +16-bit data bus
  - 20-bit address bus (was 16-bit for 8080, 64K => 1M)
  - memory organization: 64KB segments (1 MB limit)
    - CS (code segment), DS (data), SS (stack), ES (extra segment)
- Re-organize CPU into BIU (bus interface unit) and EU (execution unit) [Fig 3.1, p.74, John Uffenbeck, 2ed]
  - Allow fetch and execution simultaneously
- Internal register expanded to 16-bit
  - Allow access of low/high byte simultaneously or separately
  - Two Memory banks for odd/even-byte access

8088: PC standard



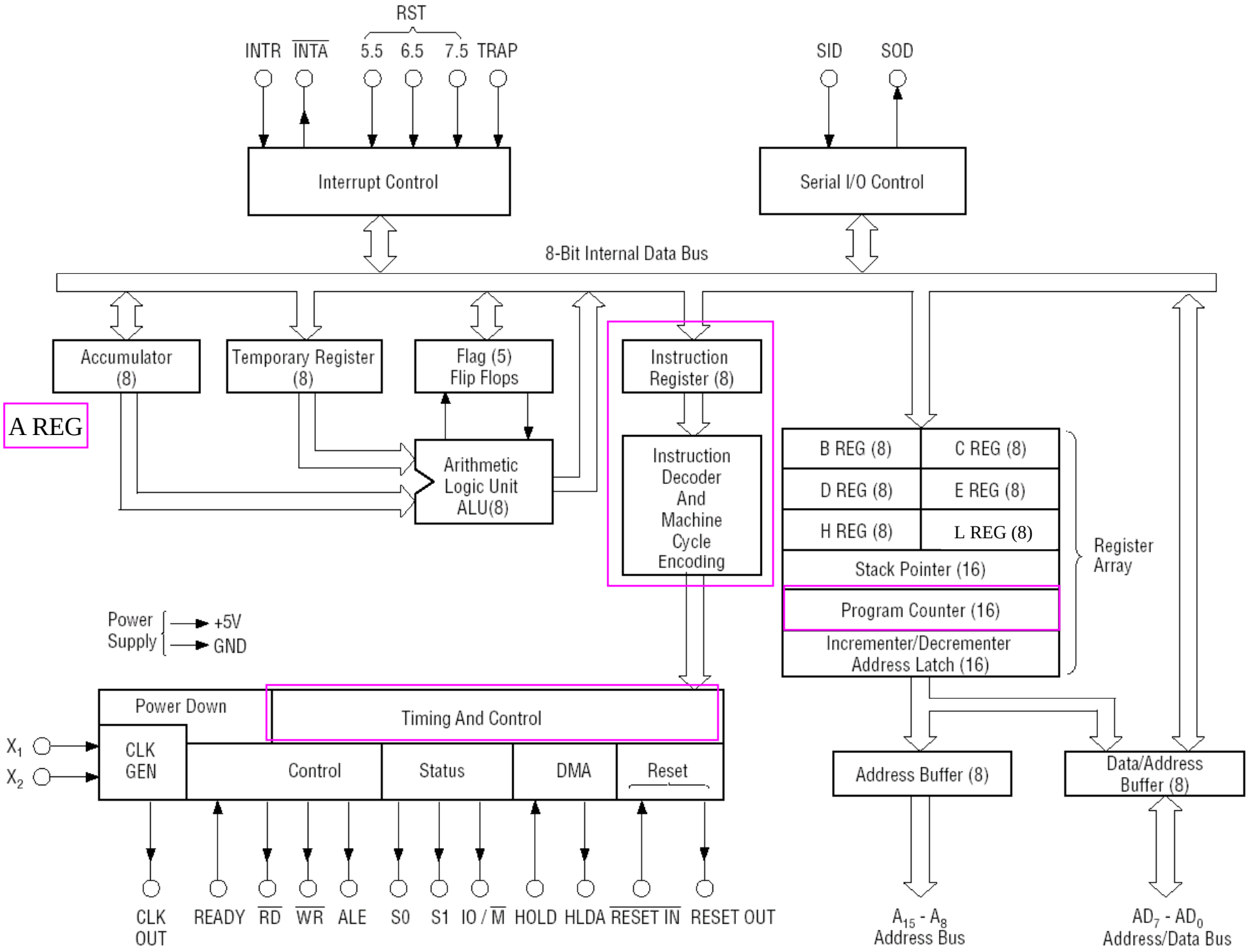
- Became available in 1979, almost identical to 8086
- 8-bit data bus: for hardware compatibility with 8080
- 16-bit internal registers and data bus (same as 8086)
- 20-bit address bus (was 16-bit for 8080)
  - BIU re-designed
- memory organization: 64KB segments (1 MB limit)
  - Two memory accesses for 16-bit data (less efficient)
  - But less costly
- 8088: used by IBM PC (1982), 16K-64K, 4.77MHz

80186, 80188: High Integration CPU

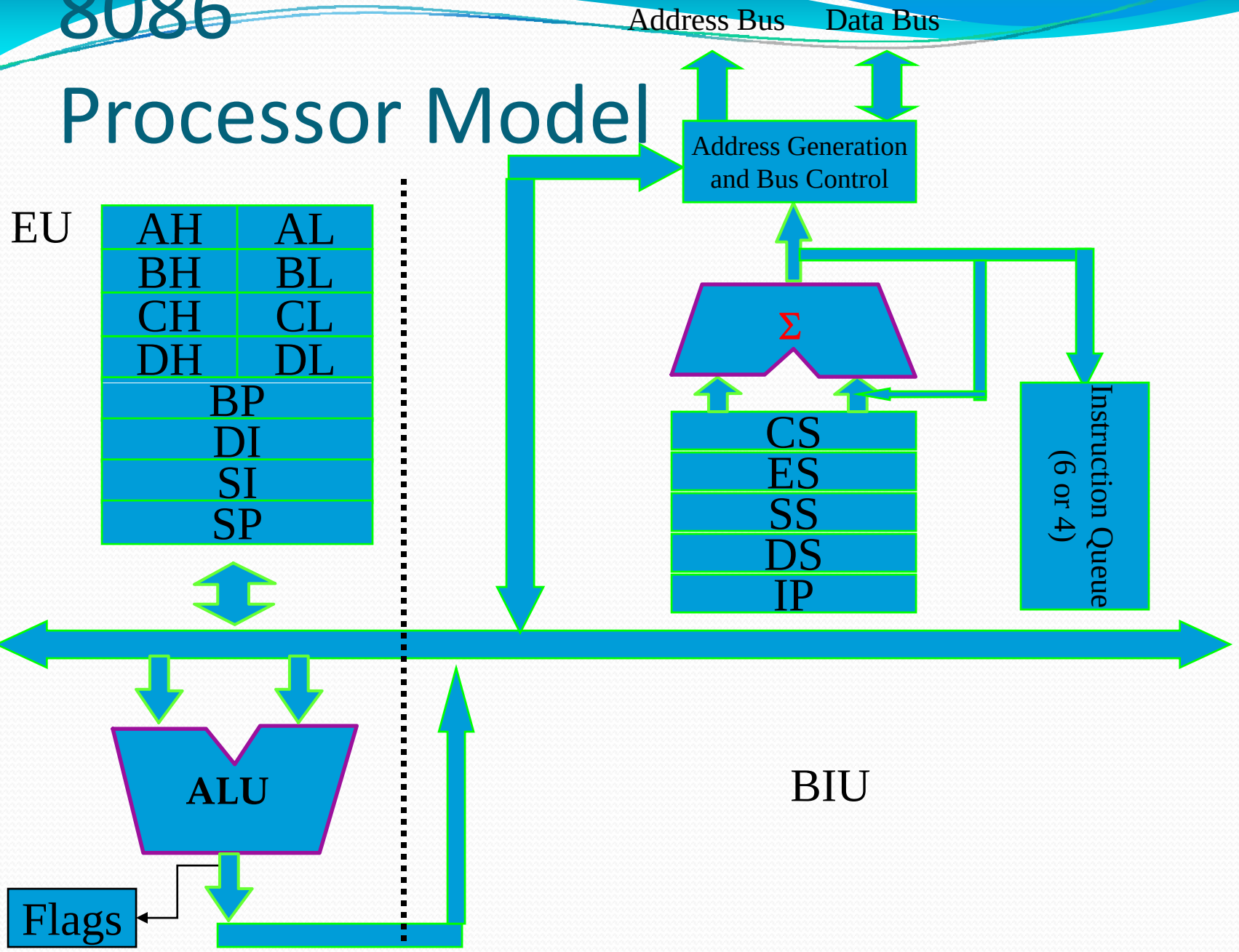
- PC system:
  - 8088 CPU + various supporting chips
    - Clock generator
    - 8251: serial IO (RS232)
    - 8253: timer/counter
    - 8255: PPI (programmable peripheral interface)
    - 8257: DMA controller
    - 8259: interrupt controller
- 80186/80188: 8086/8088 + supporting functions
  - Compatible instruction set (+ 9 new instructions)

8086 Processor Model: BIU+EU

- BIU
  - Memory & IO address generation
  - Read/Write Instructions/Data from/to Data Bus
- EU
  - Receive codes and data from BIU
    - Not connected to system buses
  - Execute instructions
  - Save results in registers, or pass to BIU to memory and IO

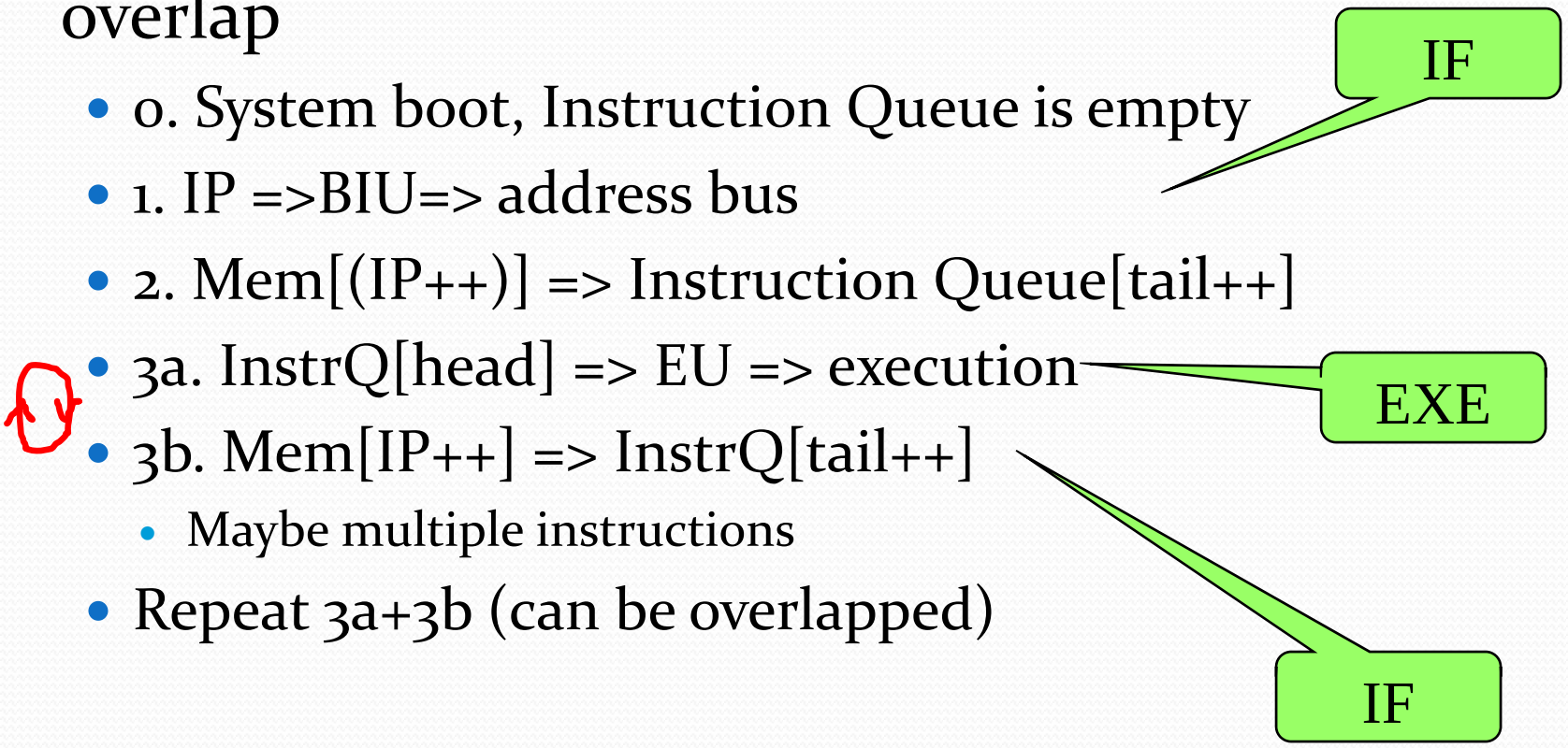

RST
INTR
INTA
5.5
6.5
7.5
TRAP
SID
SOD
Interrupt Control
Serial I/O Control
8-Bit Internal Data Bus
Accumulator (8)
Temporary Register (8)
Flag (5) Flip Flops
Instruction Register (8)
A REG
Arithmetic Logic Unit ALU(8)
Instruction Decoder And Machine Cycle Encoding
B REG (8)
C REG (8)
D REG (8)
E REG (8)
H REG (8)
L REG (8)
Stack Pointer (16)
Program Counter (16)
Incrementer/Decrementer Address Latch (16)
Register Array
Power Supply
+5V
GND
Power Down
Timing And Control
X1
X2
CLK GEN
Control
Status
DMA
Reset
Address Buffer (8)
Data/Address Buffer (8)
CLK OUT
READY
RD
WR
ALE
S0
S1
IO / M
HOLD
HLDA
RESET IN
RESET OUT
A15 - A8
Address Bus
AD7 - AD0
Address/Data Bus

8086 Processor Model



Fetch and Execution Cycle

- BIU+EU allows the fetch and execution cycle to overlap
  - 0. System boot, Instruction Queue is empty
  - 1. IP =>BIU=> address bus
  - 2. Mem[(IP++)] => Instruction Queue[tail++]
  - 3a. InstrQ[head] => EU => execution
  - 3b. Mem[IP++] => InstrQ[tail++]
    - Maybe multiple instructions
  - Repeat 3a+3b (can be overlapped)



Waiting Conditions

- BIU+EU: execute (almost) continuously without waiting, except …
- Waiting Conditions:
  - External Memory Access
  - Next Jump Instruction
  - Long & Slow Instruction

BIU: 8088 vs. 8086

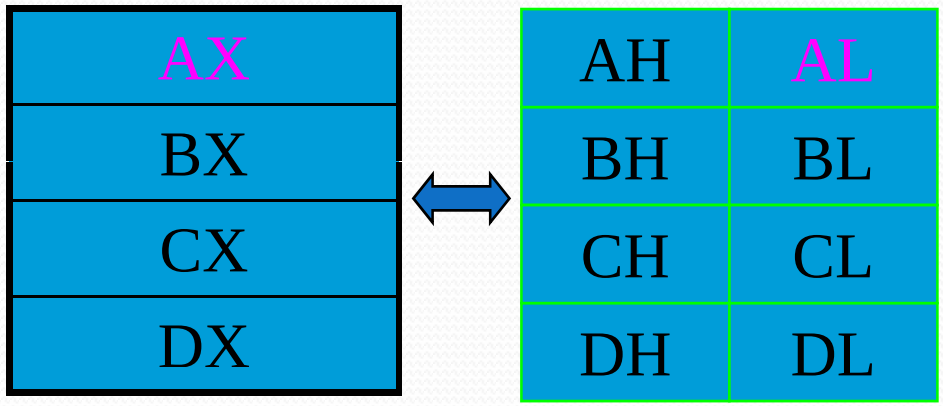
- BIU is the major difference
- 8088:
  - Register: 16-bit (same as 8086)
  - Data bus: 8-bit (vs. 16-bit/8086)
  - Instruction queue: 4 bytes (vs. 6-byte/8086)
- Only 30% slower than 8086, Why?
  - If queue is kept full, instructions are executed without interruption
  - only slightly affected by the data bus width difference: 16 vs 8-bit

8086/8088 Programming Model: Processor Features *vs*. Model

- ALU: 16-bit *vs*. 8-bit data register
  - Some operations requires 8 bits, some 16 bits
    - => Groups of 8-bit registers as 16-bit registers
    - => Memory copy: two bytes from two memory banks for 16-bit operations
- Address: 20-bit (much more memory locations)
  - With 16-bit address pointers (same as 8085)
  - Memory management: Segmented Memory
    - Divide memory into 64KB segments (16-bit addressable within each segment)
    - 16-bit Pointers: for Segment address & Offset within segment

8086/8088 Programming Model: EU Register Functions

Data Registers



16-bit registers

8-bit registers

(Register Operands)

Status and Control Flags

Flags H	Flags L

8-bit flags x 2

Segment Registers

CS
ES
SS
DS

BP
DI
SI
SP

IP

Pointers & Index Registers

(Memory Access)

8086 Programming Model: Data Registers

- Data Group: (8/16-bit registers)
  - 16-bit registers, byte/word accessible (16=8x2)
  - Data registers: Save temporary results as long as possible
    - For Efficiency: avoid costly (slow) external memory access
  - AX (= AH+AL): 16-bit Accumulator (Acc.)
    - AL = Accumulator for 8-bit operations
    - Acc: default source operand & result for arithmetic/logic operations
      - E.g., ADD AX, BX ⇔ AX := AX + BX.
  - BX (= BH+BL): typically used as a Base register
  - CX (= CH+CL): typically used as a Counter
  - DX (= DH+DL): typically used as a general Data register

8086 Status and Control Flags

- Flags: status vs. control
  - Status: indication of results
    - For conditional flow control: JNZ, JNC, …
  - Control: set or clear to control subsequent operations
- 8086:
  - status x 6 [C, P, A, Z, S, O]
  - control x 3 [TF, IF, DF]

8086 Status and Control Flags

Flag L： (Same as 8085 status register)

SF	ZF	X	AF	X	PF	X	CF

CF: Carry Flag
- CF= 0 ： No Carry (Add) or Borrow (SUB)
- CF= 1 ： high-order bit Carry/Borrow

PF: (Even) Parity Flag (even number of 1’s in low-order 8 bits of result)

AF: Aux. Carry: Carry/Borrow on bit 3 (Low nibble of AL)

ZF: Zero Flag: (1: result is zero)

SF: Sign Flag: (0: positive, 1: negative)

8086 Status and Control Flags

Flag H： (TF, IF, DF: control bits, others: status)

X	X	X	X	OF	DF	IF	TF

TF: Trap flag (single-step after next instruction; clear by single-step interrupt)

IF: Interrupt-Enable: enable maskable interrupts

DF: Direction flag: auto-decrement (1) or increment(0) index on string (block move) operations

OF: Overflow: signed result cannot be expressed within #bits in destination operand

8086 Status and Control Flags

- TF: software single step
  - Jump to trap address on each execution (if set)
- IF: INTR enable
  - INT, ISR, INTV, IRET
- DF: block move (string) operation direction
  - Auto-increment or auto-decrement

FIGURE 3-5 8086 flag word, DF, IF, and TF can be set or reset to control the operation of the processor. The remaining flags are status indicators. Bits marked X are undefined.

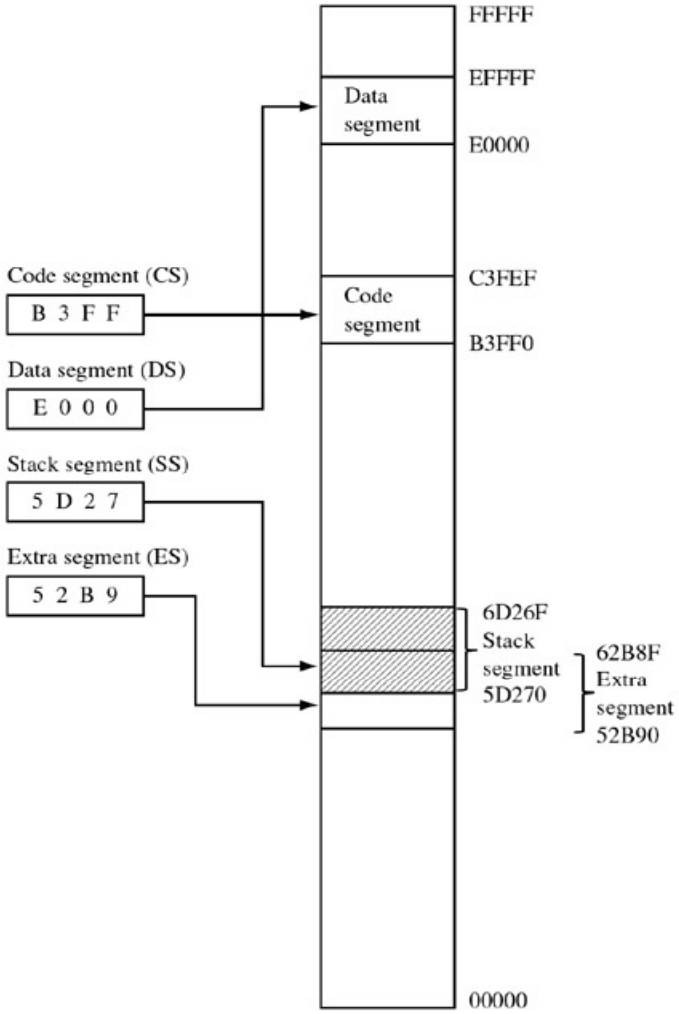
Bit Position	Name	Function
0	CF	Carry flag: Set on high-order bit carry or borrow; cleared otherwise
2	PF	Parity flag: Set if low-order 8 bits of result contain an even number of 1 bits; cleared otherwise
4	AF	Set on carry from or borrow to the low-order 4 bits of AL; cleared otherwise
6	ZF	Zero flag: Set if result is zero; cleared otherwise
7	SF	Sign flag: Set equal to high-order bit of result (0 is positive, 1 is negative)
8	TF	Single-step flag: Once set, a single-step interrupt occurs after the next instruction executes; TF is cleared by the single-step interrupt
9	IF	Interrupt-enable flag: When set, maskable interrupts will cause the CPU to transfer control to an interrupt vector-specified location
10	DF	Direction flag: Causes string instructions to auto-decrement the appropriate index register when set; clearing DF causes auto-increment
11	OF	Overflow flag: Set if the signed result cannot be expressed within the number of bits in the destination operand; cleared otherwise

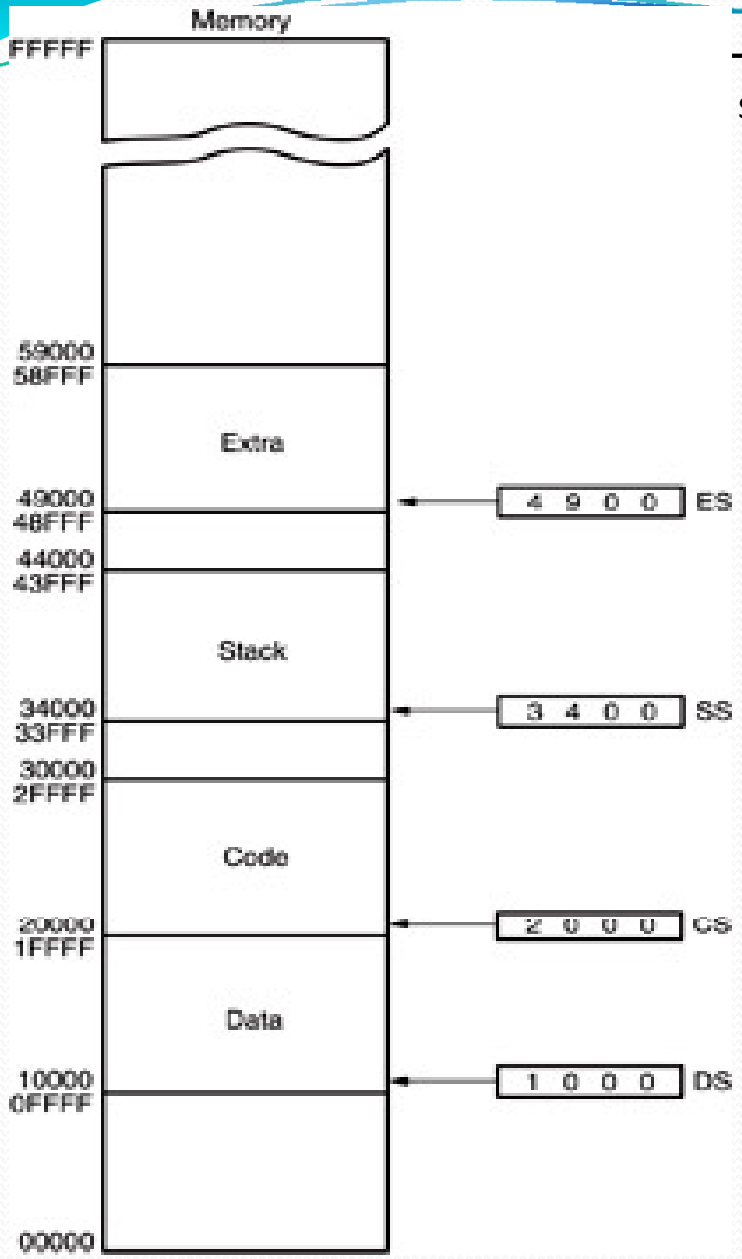
8086 Programming Model: Segment Registers

- Segment Group: divide memory into
  - CS: Code Segment [64KB for each segment]
  - DS: Data Segment
  - ES: Extra Segment
  - SS: Stack Segment
- Segment Registers: CS, DS, ES, SS
  - Save Base addresses to particular segments
  - SEG(16-bit):OFFSET(16-bit) is used by BIU to calculate (20-bit) physical memory address

[DS:0] = byte 0
(the 1-st byte)
in Data Segment
0 = offset to begin-of-DS

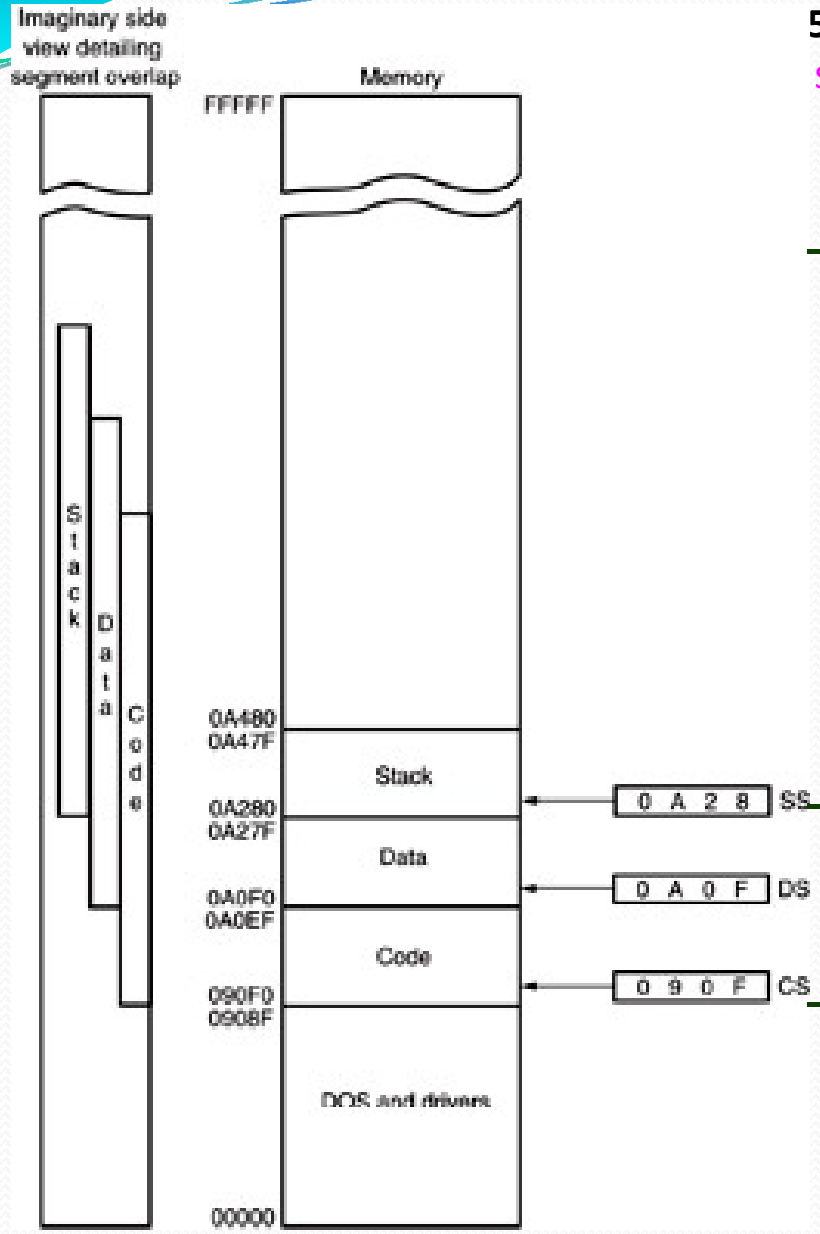
FIGURE 3-9 The 8086 divides its 1 MB of memory address space into four segments, the data, code, stack, and extra segments. The four segment registers DS, CS, SS, and ES point to location 0 of the current segment. In this example, the stack and extra segments are partially overlapped. (From J. Uffenbeck, *Microcomputers and Microprocessors: The 8080, 8085, and Z-80*, I





-4 A memory system showing the placement of four segments.

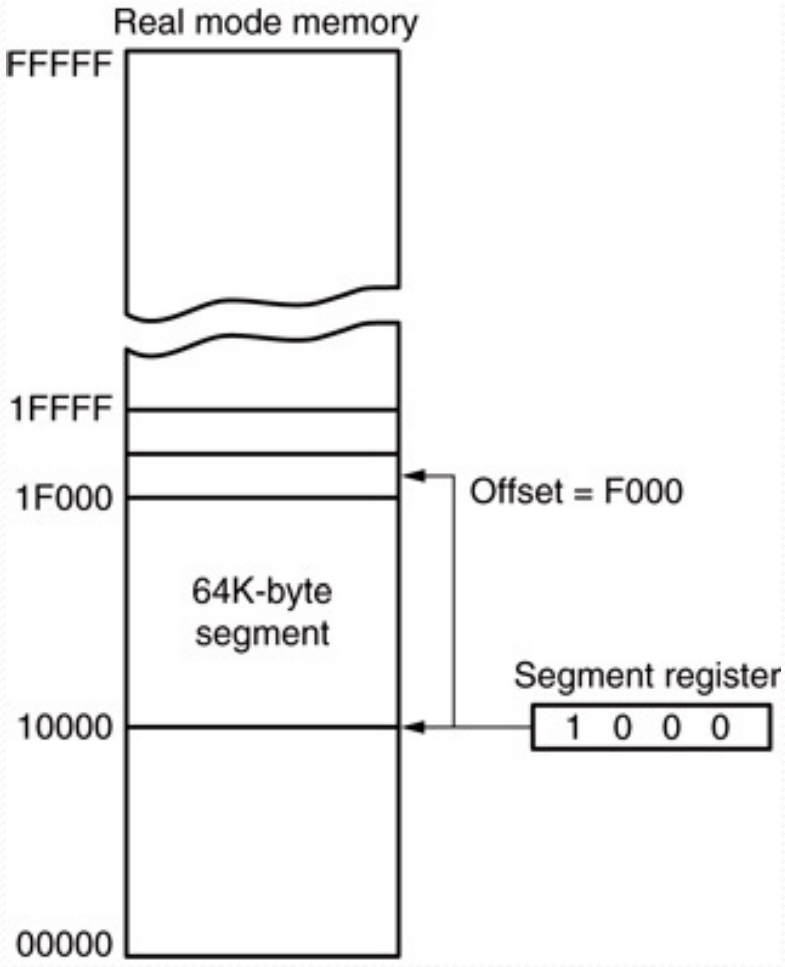
- think of segments as windows that can be moved over any area of memory to access data or code
- a program can have more than four or six segments,
  - but only access four or six segments at a time



5 An application program containing a code, stack segment loaded into a DOS system

- a program placed in memory by DOS is loaded in the TPA at the first available area of memory above drivers and other TPA programs
- area is indicated by a **free-pointer** maintained by DOS
- program loading is handled automatically by the **program loader** within DOS

Figure 2–3 The 8086 (or real mode) memory-addressing scheme, using a segment address plus an offset.



– this shows a memory segment beginning at 10000H, ending at location 1FFFFH
  - 64K bytes in length

– also shows how an offset address, called a **displacement**, of F000H selects location 1F000H in the memory

8086 Programming Model: Index Registers

- Pointer/Index Group: (16-bit) as memory pointers
  - IP: Instruction Pointer ⇔ CS (point to CS by default)
    - (next instruction to be fetched by BIU; physically part of BIU)
  - SI: Source Index ⇔ DS
  - DI: Destination Index ⇔ ES
  - SP: Stack Pointer ⇔ SS
- Index Registers: IP, SI, DI, SP
  - Save Index (or offset) or Pointer to a Base address
    - E.g., MOV AH, [SI] ; AH := (*DS:SI), // Mem. Addr. in SI

Segmented Memory

- Memory Organization: Linear vs. Segmented
  - Linear Addressing: (MC68K, i8085)
    - The entire memory is regarded as a whole
      - Specify *absolute* addresses in instructions
    - The entire memory space is available all the time
  - Segmented Addressing: (ix86)
    - Memory is divided into segments
      - Specify an address as *offset* relative to *segment base* address
      - Programs use offsets as logical address, independent of where segments are located (relocatable)
    - Process is limited to access designated segments at a given time

Segment Registers

- 8086: 1M, divided into 64K (2^16) memory segments
  - 16-bit offset/logical address (relative to segment base address)
- 4 active segments, pointed to by
  - CS (program codes), DS (data for program), ES (extra/shared data), SS (stack or subroutine/ISR return addresses) segment registers
- 8085: 64K x 1, for program and data
  - Stack contents may overwrite data and code
  - Limited program code size

8086 Programming Model: Segment Registers

- Segment Group: divide memory into
  - CS: Code Segment [64KB for each segment]
  - DS: Data Segment
  - ES: Extra Segment
  - SS: Stack Segment
- Segment Registers: CS, DS, ES, SS
  - Save Base addresses to particular segments
  - SEG(16-bit):OFFSET(16-bit) is used by BIU to calculate (20-bit) physical memory address

[DS:0] = byte 0
(the 1-st byte)
in Data Segment
0 = offset to begin-of-DS

Logical and Physical Addresses

- Physical Address: 20-bit
- Index/segment registers: 16-bit
  - Logical address in index registers: 16-bit
  - Base address in segment registers: 16-bit+0000_2
    - 16-byte segment boundaries
    - Can define a segment at absolute addresses: 16N+0.
  - Address Translation:
    - Phys-address = Base*16+Index (or offset)
    - E.g., CS:IP, DS:2345H, …

Default Segment Registers

- The default segment register depends on the instruction being used
  - MOV [BP], AL ; AL := *(SS:BP)
  - Next instruction ⇔ CS:IP
- Alternative segment:
  - Default segment can be changed using segment override operator (for some memory reference types)

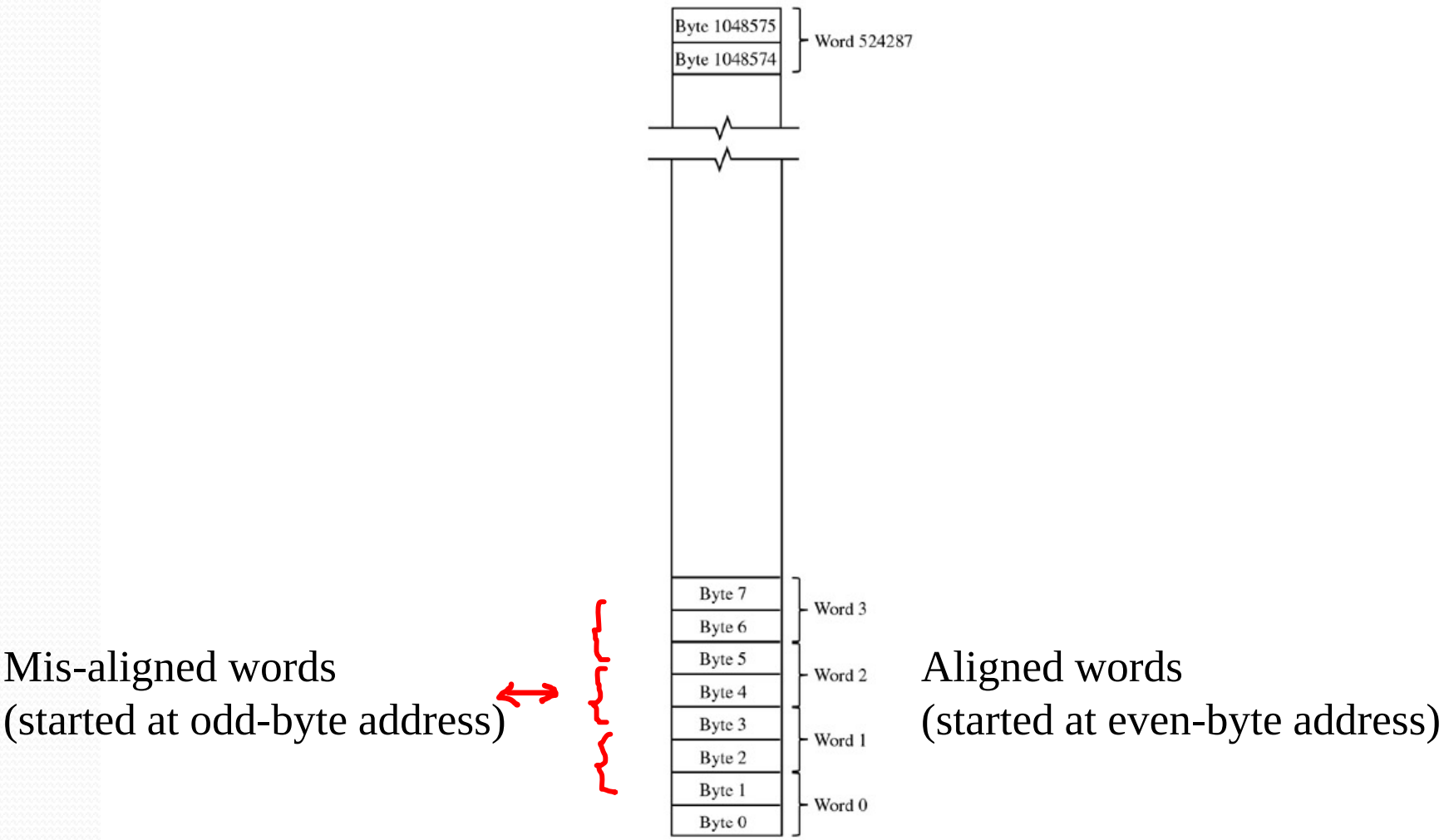
Default Segment Registers

Type of reference	Default segment	Alternative segment	Offset (logical address)
Instruction fetch	CS	-	IP
Stack operations	SS	-	SP
General data	DS	CS,ES,SS	Effective address
String source	DS	CS,ES,SS	SI
String destination	ES	-	DI
BX used as pointer	DS	CS,ES,SS	Effective address
BP used as pointer	SS	CS,ES,DS	Effective address

Segmented Memory

- Advantages
  - CS+DS1/DS2/…: different DS's for one program
  - CS1 => CS2:
    - re-allocatable codes for task switching
    - Run at any location (if no reference to physical address)
- Disadvantages
  - Complex hardware: requires two registers (e.g., DS:SI)
  - Limited segment size for a program
    - 64K or, if larger, switching between 64K's
    - 386~: large segment up to 4G (flat mode, disabling segmented memory)

FIGURE 3-6 The memory space of the 8086 consists of 1,048,576 bytes or 524,288 16-bit words.



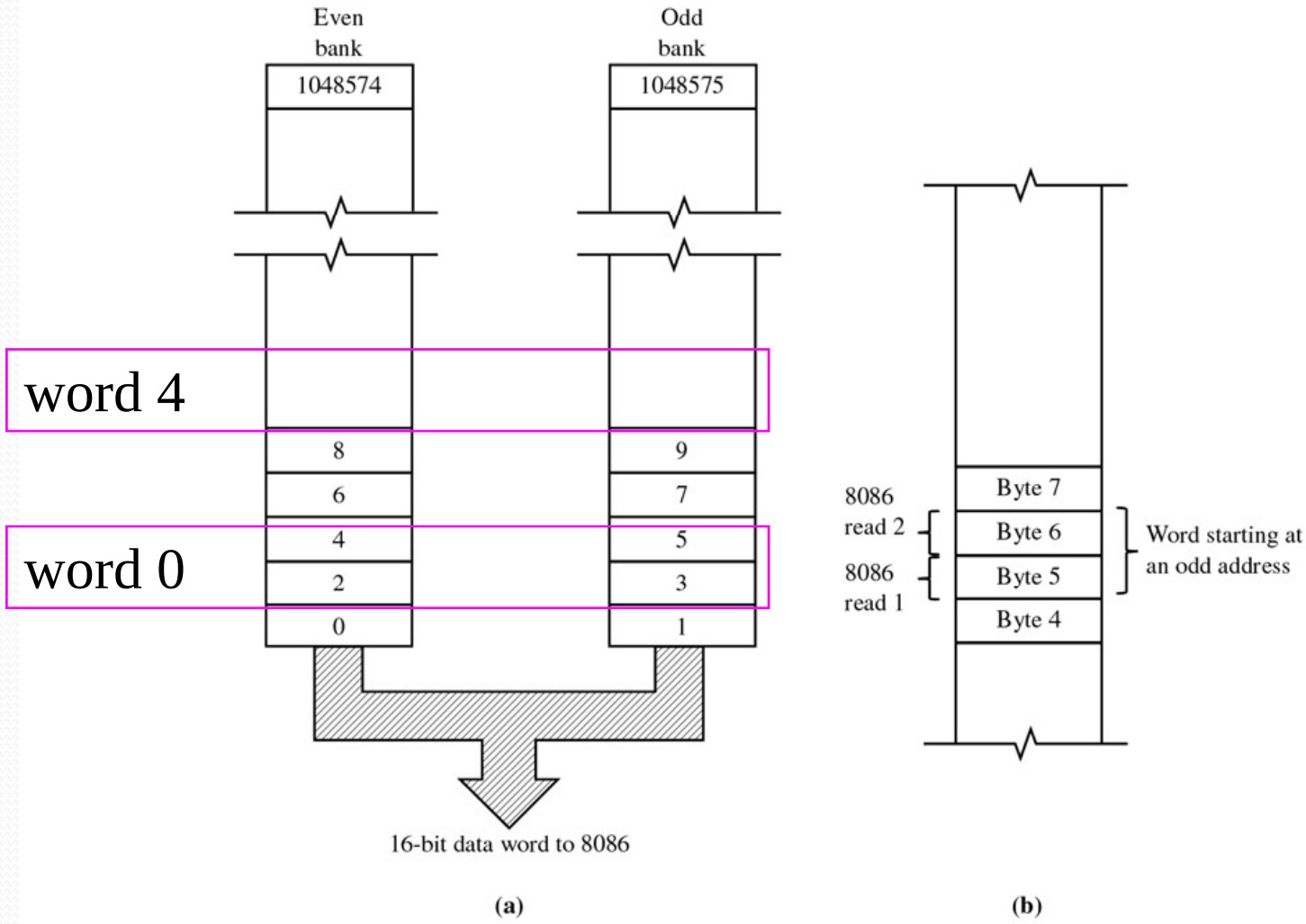
8086 Memory Organization: Memory Banks (1)

- 2**20 addresses can be arranged as:
  - 2**20 (8-bit) *bytes* in continuous locations, or
  - 2**19 16-bit aligned *words* (in parallel banks)
    - Plus 2**19-1 mis-aligned words
  - Not a single way, but …
- Memory organization & interfacing should reflects the access behavior of the processor in the most natural way. Two types of operations:
  - Byte: I/O, character data (e.g., ASCII)
  - Words: large 16-bit integers arithmetic operations

8086 Memory Organization: Memory Banks (2)

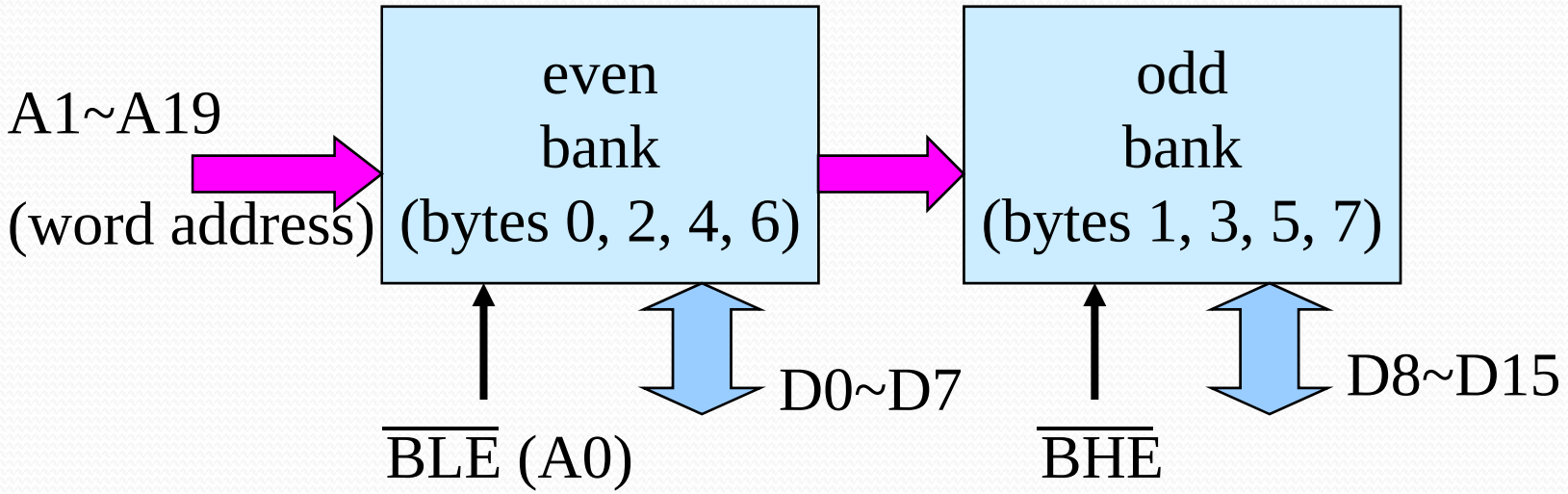
- Even and Odd Memory Banks
  - 16-bit data bus⇔ two-byte / two one-byte access
  - Allows processor to work on bytes or on (16-bit) words
    - IO operations are normally conducted in bytes
    - Arithmetic operations may use large 16-bit words
  - Can handle odd-length instructions
    - Single byte instructions
    - Multiple byte (and very long) instructions

FIGURE 3-7 (a) By reading from an even-addressed bank and an odd-addressed bank the 8086 can read two bytes from memory simultaneously. (b) If the 16-bit word begins at an odd address, the 8086 will require two memory read or write cycles.



Even/Odd Banks, Word Addresses *vs*. Byte Addresses

Word address (with 2 banks) = Byte address div 2
(= discarding the least significant address bit)
e.g., bytes xxxx…zzz0 and xxxx…zzz1 belong to word xxxx…zzz (x,z: 0's or 1's)



8086 uses A19~A1 to specify a word address, and uses A0 (also know as BLE) and BHE to select individual bytes in the word, or select both (the word).

8086 Memory Organization: Memory Banks (3a)

- Memory Banks
  - Can read 16-bit data simultaneously
    - ⇔One from odd-addressed byte, another from even-addressed byte
    - ⇔Need TWO memory banks in parallel
  - Byte Access = Word Access + Byte Enable
    - Select byte(s) in accessed word
    - BLE: Byte/Bank Low Enable
      - Labeled as A0 in 8086
      - Confusing: No A0, A1 in later x86 CPUs
    - BHE: Byte/Bank High Enable

8086 Memory Organization: Memory Banks (3b)

- Address Decoding (difference with 8085/88)
  - 8085/8088: (byte-oriented memory access)
    - A0~A15: issue *byte* addresses to address individual bytes
    - A0~An-1 (low order n address bits): connected to A0~An-1 of memory chips with n address pins
    - An~A15 (high order address bits): for chip selection (CS)
  - 8086: (word-oriented memory access)
    - A1~A19: issue *word* addresses to address words (= 2 parallel bytes)
      - A1~An: Connected to A0~An-1 of two parallel memory banks
      - An+1~A19: for chip selection (CS)
    - A0(BLE#) and BHE# control line:
      - to select a byte from even/odd bank for byte-based operations and operations involving misaligned operands, or
      - select both bytes for word-based operations.

FIGURE 7-22 64K x 8 8088(/85) SRAM interface. Only a single memory chip is required.

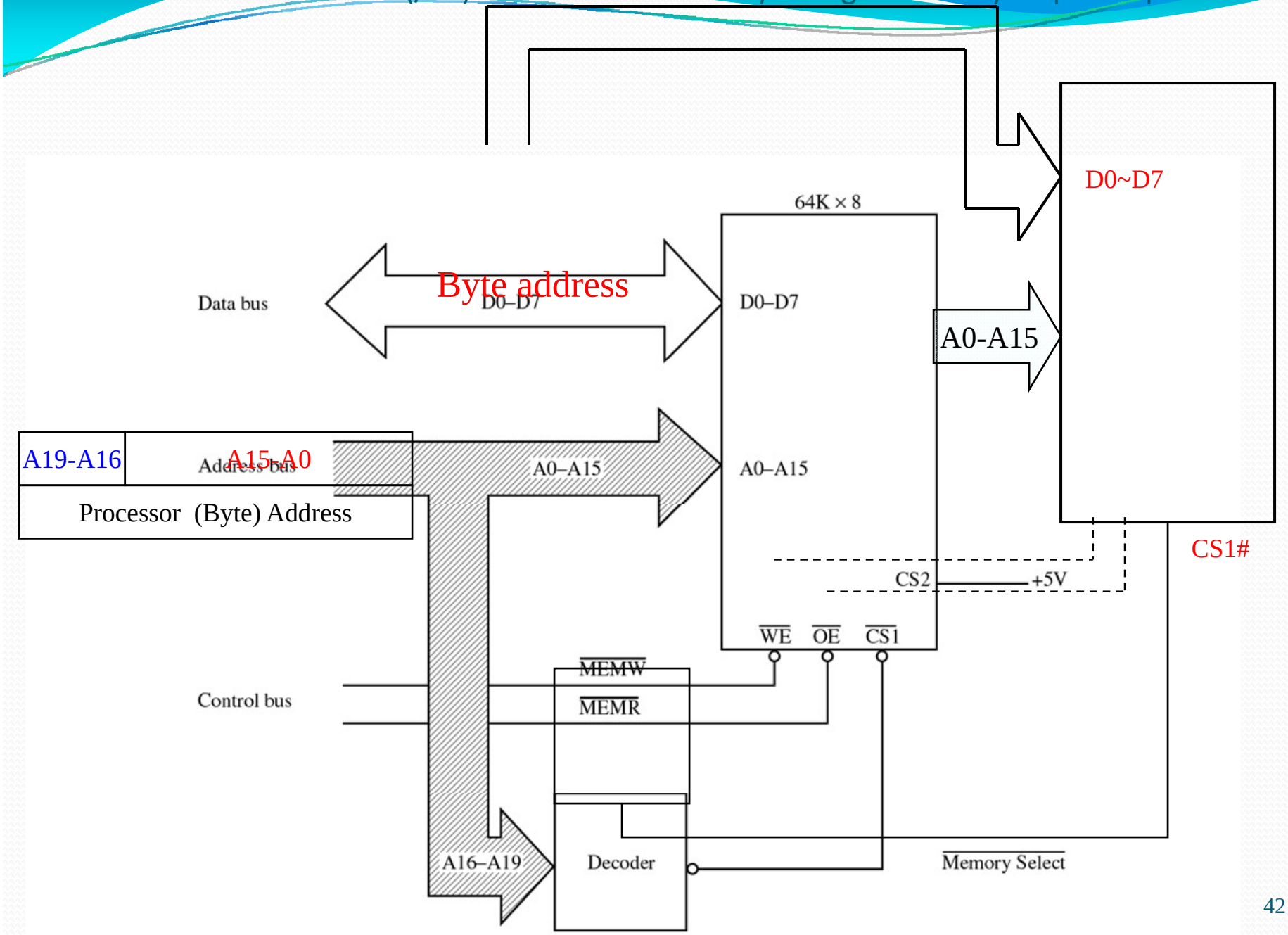
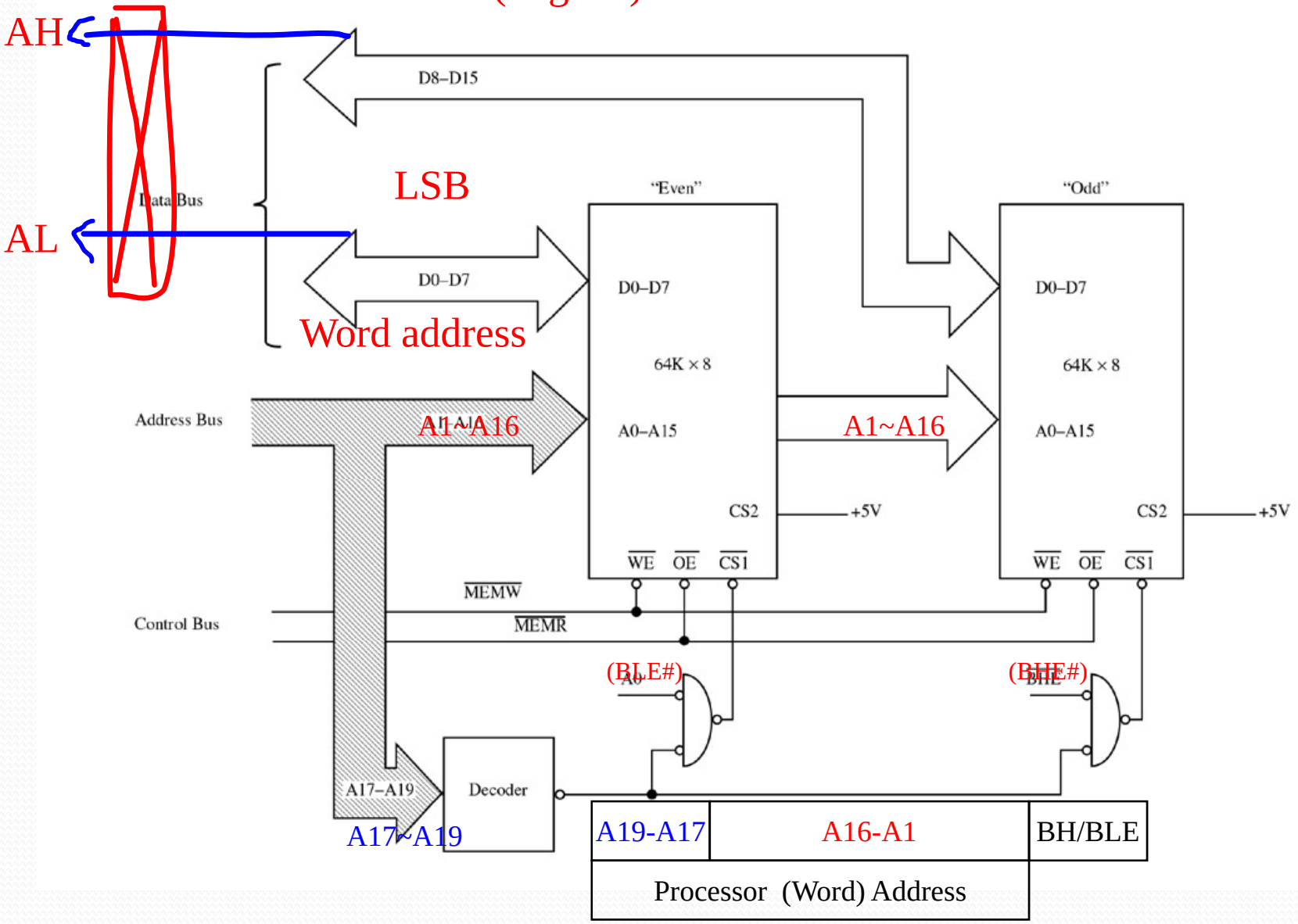


FIGURE 7-26 128K x 8 8086 SRAM interface.



Memory Organization: Alignment

Addr	data
0000	H1
0001	L1
0002	H2
0003	L2
0004	H3
0005	L3

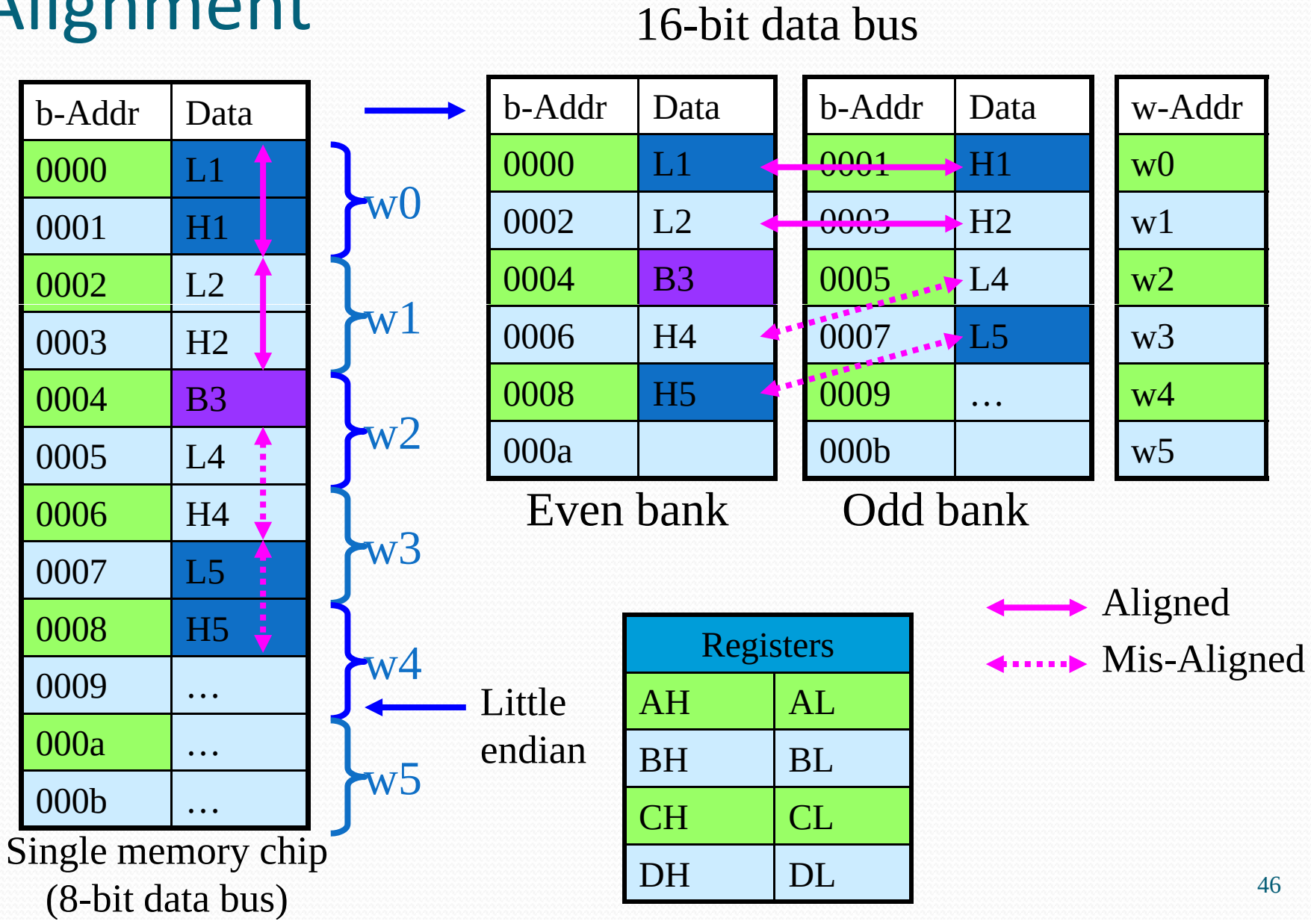
- Endianess:
  - Single way to model multi-byte CPU *register*
    - AX ⇔ AH+AL (high order byte in AH, low order byte in AL)
  - Two ways to store operands in *memory*
- Big-endian CPU: (IBM370, M68*, Sparc)
  - High-order-byte-first (HOBF)
  - Maps highest-order byte of internal register⇔lowest (1st) memory byte address
  - Operand address == address of 1st memory byte (MSB)
    - MOV R1, [N] ⇔ N: addr. of 1st byte in memory & MSB of register
    - N, N+1, N+2, …: addresses of all bytes, MSB at addr [N]

Memory Organization: Alignment

Addr	data
0000	L1
0001	H1
0002	L2
0003	H2
0004	L3
0005	H3

- Little-endian CPU: (DEC, Intel)
  - Low-order-byte-first (LOBF)
  - Maps lowest-order byte of register ⇔1st memory byte
  - Operand address == address of 1st memory byte (LSB)
    - MOV AX, [N] ⇔N: addr of 1st byte in memory & LSB of register
    - AL⇔[N], AH⇔[N+1] (addr [N], +1, +2, ...: low-to-high order)
- Configurable CPU:
  - Can switch between Big/Little-endian, or
  - Provide instructions which convert 16-/32-bit data between two byte ordering (80486)

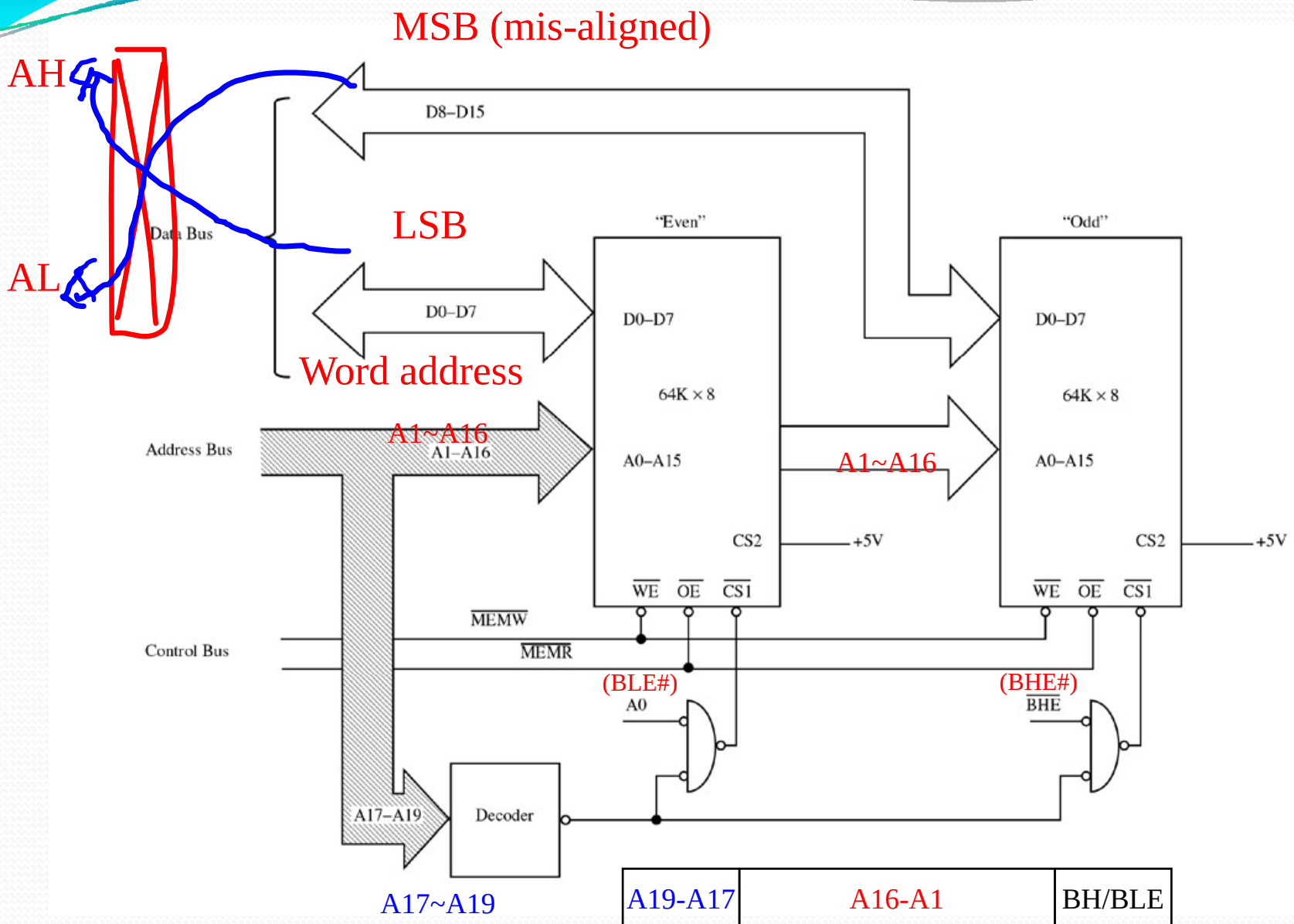
Memory Organization: Alignment



8086 Memory Organization

- Aligned operand:
  - Operand aligned at even-byte (word/dword) boundaries
  - Allows single access to read/write one operand
- Mis-aligned words:
  - Word operand does not start at even address
    - Through internal shift/swap mechanism, if necessary (e.g., load mis-aligned MSB/LSB to AH/AL)
  - Need 2 read cycles to read/write the word (8086)
    - Issues two addresses to access the two even-aligned words containing the operand in order to access the operand
    - slower but transparent to programmer

FIGURE 7-26 128K x 8 8086 SRAM interface.



A19-A17	A16-A1	BH/BLE
Processor (Word) Address		

8086 Memory Organization

- 8088: 8-bit data bus (same as 8085)
  - Always 2 cycles for word-based operations
    - Aligned or not
    - Slower word operations
  - because of 8-bit external data bus
    - Use single memory bank to store multi-byte operands (like i8085)

8086 Memory Map

- Memory Map: How memory space is allocated
  - ROM Area: boot, BIOS
  - RAM: OS/User Apps & data
  - Unused
  - Reserved: for future hardware/software uses
  - Dedicated: for specific system interrupt and rest functions, etc.

386 Processor

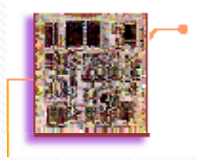
Review 286/386 Protected Modes

Processor Model: BIU+CPU+MMU

Programming Model: GPR + SPR (Segments vs. Paging) + PL

80286 (Review)

- First with Protection Mode
- Review of 286 Protected Mode … Next



80286

- Became available in 1982
- used in IBM AT computer (1984)
  - 16-bit data bus
  - 24-bit address bus (16 MB)
    - (vs. 20-bit/1M 8086)
- clock speed 25% faster than 8088, throughput 5 times greater than 8088

80286: Real vs. Protected Modes

- Larger address space: 24-bit address bus
  - Real Mode vs. Protected Mode
- Real Mode: (8086 Mode)
  - Power on default mode
  - Function like a 8086: use 20-bit least significant address lines (1M)
  - Software compatible with 8086, 186
  - 16 new instructions (for Protected Mode management)
  - Faster 286: redesigned processor, plus higher clock rate (6-8MHz)

80286: Real vs. Protected Modes

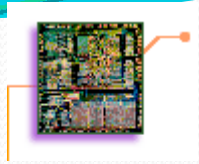
- Protected Mode:
  - *Multi-program* environment
  - Each program has a predetermined amount of memory
  - Addressed via segment selector (physical addresses invisible): 16M addressable
    - Easy program switching
  - "Protected mode": Multiple programs loaded at the same time (within their respective segments), protected from read/write by each other; a program running in another segment cannot Read/Write other segments

80286: Real vs. Protected Modes

- Protected Mode:
  - Cannot be switch back to real mode to avoid illegal access by switching back and forth between modes
- A faster 8086 only?
  - MS-DOS requires that all programs be run in Real Mode

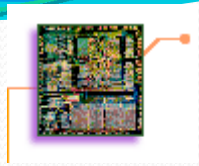
80386 Model

- Refine 286 Protect Mode
  - Real & Protected Modes
- Expand to 32-bit registers
- New Virtual 8086 Mode
- Components: BIU, CPU, MMU



80386DX (aka. 80386)

- available in 1985, a major redesign of 86/286
  - Compatibility commitment through 2000
- 32-bit data and address buses (4 GB memory)
  - Real Address Mode: 1M visible, 286 real mode
  - Protected Virtual Address Mode:
    - On board MMU
    - Segmented tasks of 1byte to 4G bytes
      - Segment base, limit, attributes defined by a descriptor register
    - Page swapping: 4K pages, up to 64TB virtual memory space
    - Windows, OS/2, Unix/Linux



80386DX (aka. 80386)

- Virtual 8086 mode (a special Protected mode feature): permitted multiple 8086 virtual machines-multitasking (similar to real mode)
  - Windows (multiple MSDOS's)
- Clock rate:
  - max. 40MHz, 2 pulses per R/W bus cycle
  - External memory cache to avoid wait
    - Fast SRAM
    - 93% hit rate with 64K cache
- Compatible instructions (14 new)

80386: Real vs. Protected Modes

- Larger address space: 32-bit address bus (4G)
  - Real Mode vs. Protected Mode (refined from 286)
- Real Mode: (8086 Mode)
  - Power on default mode
  - Function like a 8086: (1) use only 20-bit least significant address lines (1M) (2) segmented memory retained (64K)
  - Software compatible with 286
- New Real Mode Features:
  - access to 32-bit register set
  - two new segments: F, G

80386: Real vs. Protected Modes

- Protected Mode:
  - new addressing mechanism
    - (in contrast to that of real mode)
  - supports protection levels (PLs)
  - Segment size: 1 to 4G (not fixed size 64K)
  - Segment register (16-bit): as POINTER to a descriptor table
    - NOT as BASE address of a segment
    - 13-bit index to table, 1-bit local/global flag, 2-bit: RPL

80386: Real vs. Protected Modes

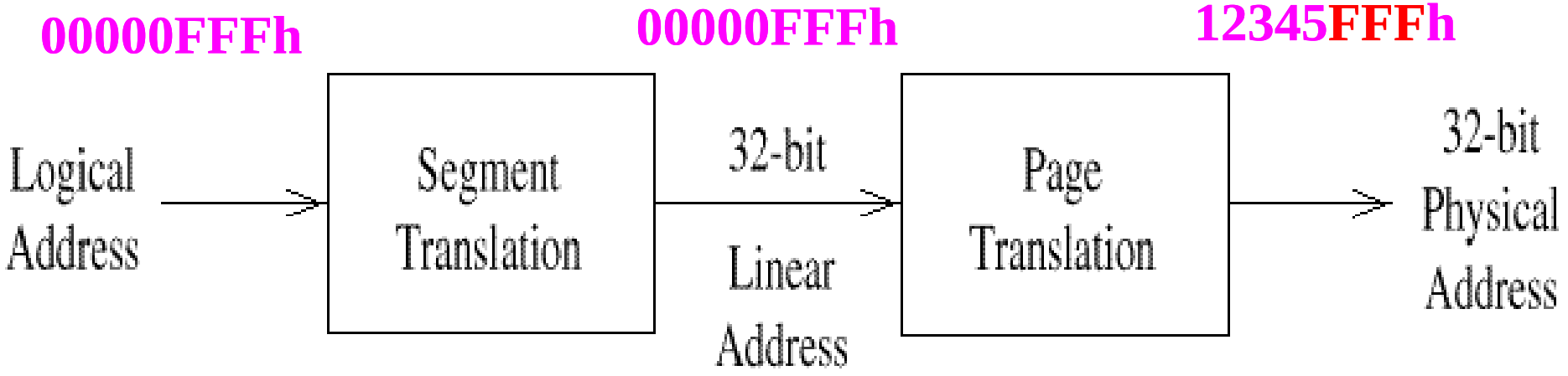
- Protected Mode: (cont.)
  - descriptor table: (8 byte per entry)
    - 32-bit base address of segment
    - segment size (20-bit):
      - in byte (max=1M, G flag=0) or in 4k-page (max=4G, G=1)
    - access rights (and status & control information)
  - memory address
    - = base address (in table) + offset (in instruction)

80386: Real vs. Protected Modes

- Protected Mode: (cont.)
  - Paging mechanism for virtual memory:
    - map 32-bit linear address (base+offset) => physical address & page frame address
    - ⇔(4K page frames in system memory)
    - ⇔up to 64TB of virtual memory
  - Paging mechanism can be turned off

Protected Mode Architecture

- In protected mode, Pentium supports
  - * More sophisticated segmentation
    - » Segmentation can be made invisible (flat model)
  - * Paging for virtual memory
    - » Paging can be turned off



80386: Real vs. Protected Modes

- Protected Mode: (cont.)
  - Protection mechanism:
    - tasks/data/instructions are assigned a privilege level (PL)
    - tasks running at lower PL cannot access tasks or data segments at a higher PL
  - OS runs multiple programs that are protected from the others

80386: Real vs. Protected Modes

- Two Ways to Run 8086 Programs:
  - Real Mode
  - Virtual 8086 Mode
- Virtual 8086 Mode:
  - runs multiple 8086 +other 386 (protected mode) programs independently
  - each task sees 1 MB (mapped via paging to anywhere in 4GB space)
  - running V8086+ Protected mode simultaneously
  - 8086 tasks is assigned the lowest privilege level, cannot access programs/data in other segments

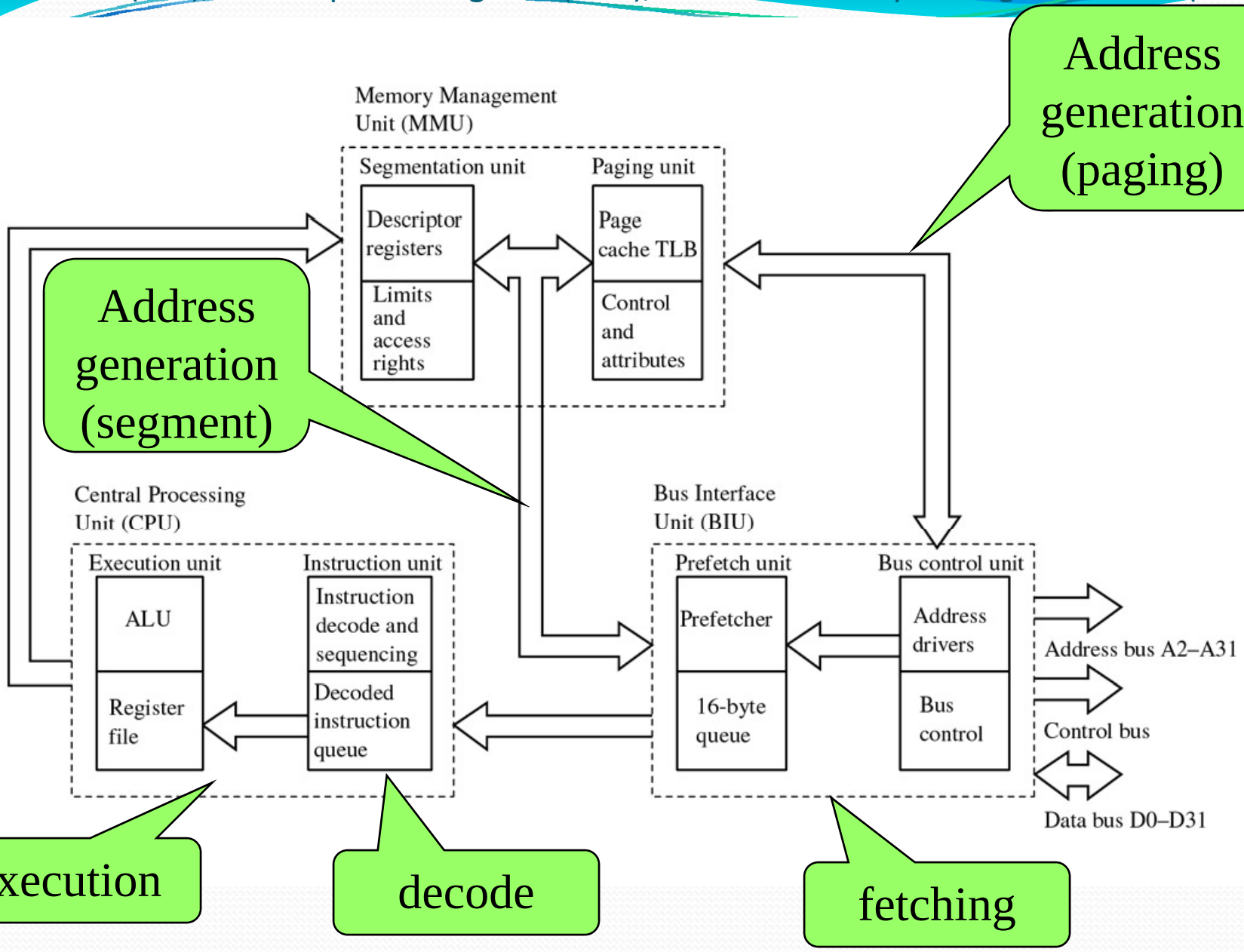
386 Processor

Review 286/386 Protected Modes

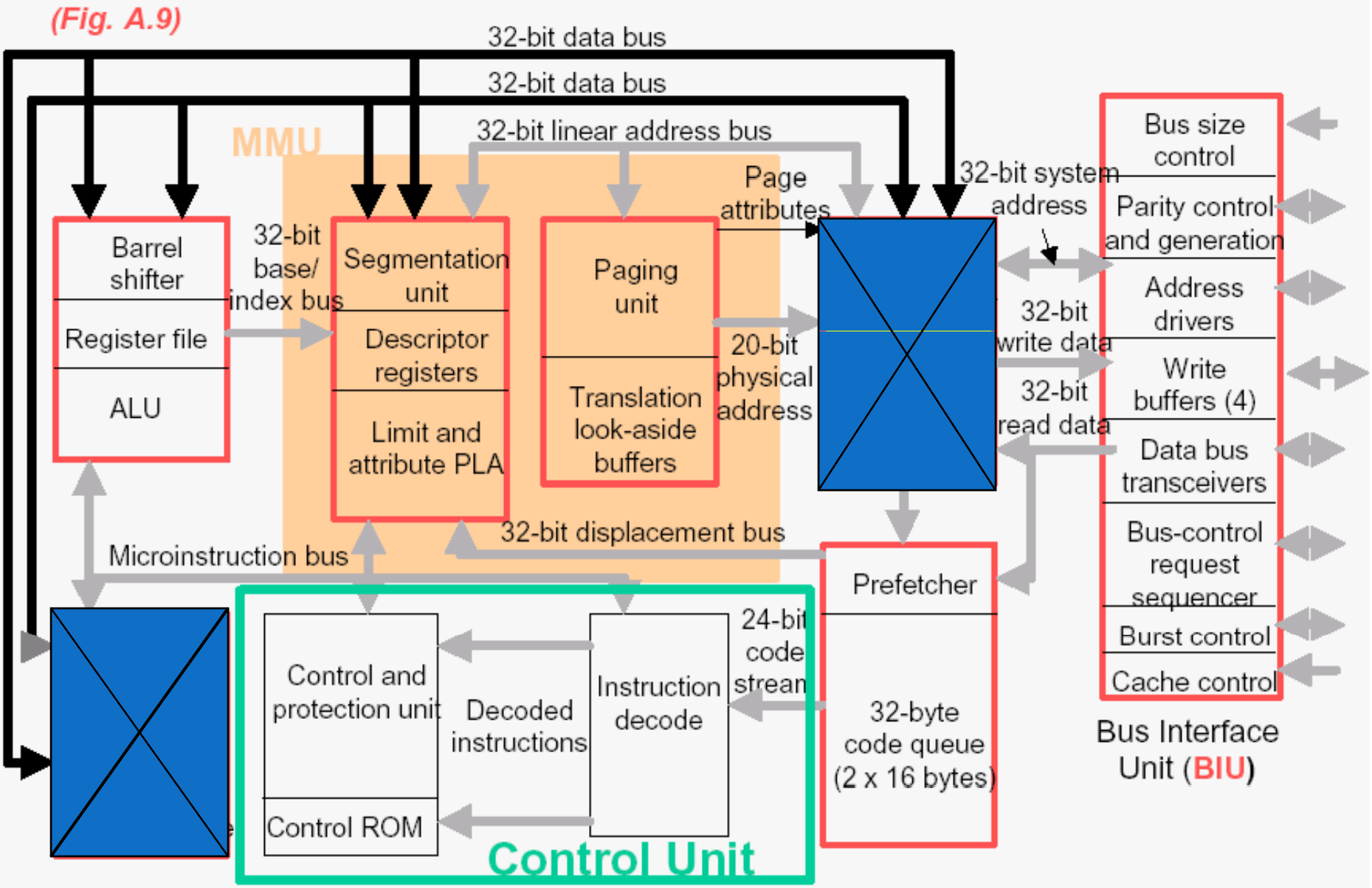
Processor Model: BIU+CPU+MMU

Programming Model: GPR + SPR (Segments vs. Paging) + PL

FIGURE 3-11 The processor model for the 80386 microprocessor consists of the bus interface unit (BIU), central processing unit (CPU), and the memory management unit (MMU).



80386 Processor Model

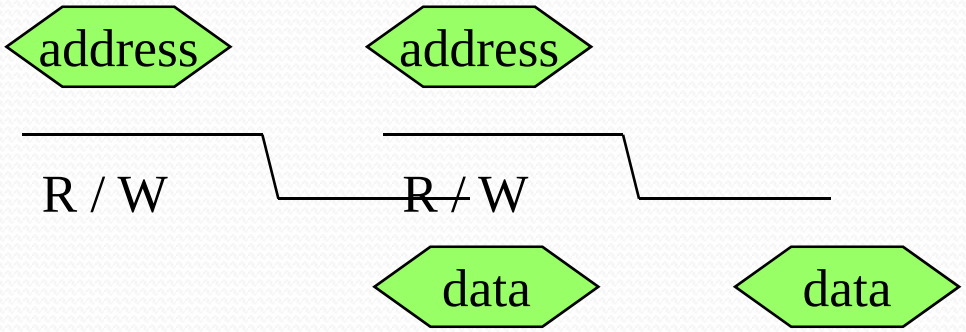


80386 Processor Model: BIU+CPU+MMU

- BIU
  - control 32-bit address and data buses
  - keep instruction queue full (16 bytes)
  - New features: address pipelining & dynamic bus sizing

80386 Processor Model: BIU+CPU+MMU

- Address Pipelining
  - address of next memory location is output halfway through current bus cycle
    - Gives external memory more address decode time
    - slower memory chip is OK
    - easier to keep up with faster (2 CLK) bus cycle of 386



80386 Processor Model: BIU

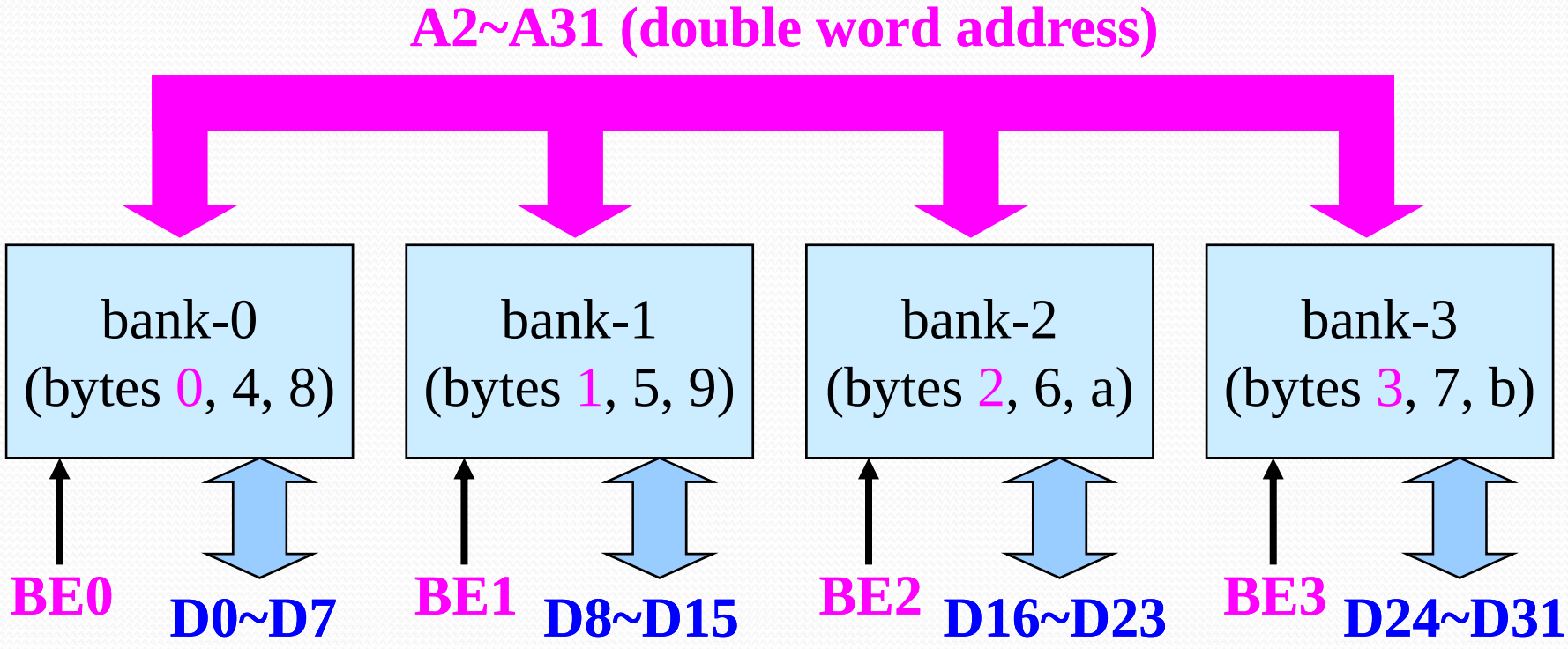
- Dynamic Data Bus Sizing
  - Switch between 16 ⇔ 32-bit *data* bus on the fly
    - accommodate to external 16-bit memory cards or IO devices
  - Adjust bus timing to use only the least significant 16 bits

80386 Processor Model: BIU

- External Memory Banks
  - 4 memory banks (4x8=32bits)
  - A2~A31: issues double word (32bit) addresses
  - BE0-BE3 for bank selection (there is no A0~A1)
    - access byte or word or double word
    - aligned operands: 1 bus cycle
    - mis-aligned (addr not at 4N, i.e., %4 !=0): 2 bus cycles

80386 Processor Model: BIU

- External Memory Banks



80386 Processor Model: CPU

- CPU=IU (instruction) +EU (execution)
  - fetching & execution can overlap
- IU:
  - retrieval instructions from queue
  - Decode instruction
  - store in decoded queue
- EU: ALU + registers (32-bit)
  - execute decoded instructions

80386 Processor Model: MMU=Segmentation+Paging Units

- Segmentation unit
  - Real mode: generate the 20-bit physical address
  - Protected mode: store base/size/rights in descriptor registers
    - cache descriptor tables in RAM
    - faster switching between tasks
- Paging Unit
  - determines physical addresses associated with active segments (divided into 4K pages)
  - virtual memory support to allow larger programs

80386 Programming Model: GPR

- General Purpose Registers (GPR)
  - Data & Addresses Groups
  - Status & Control Flags
  - Segment Group

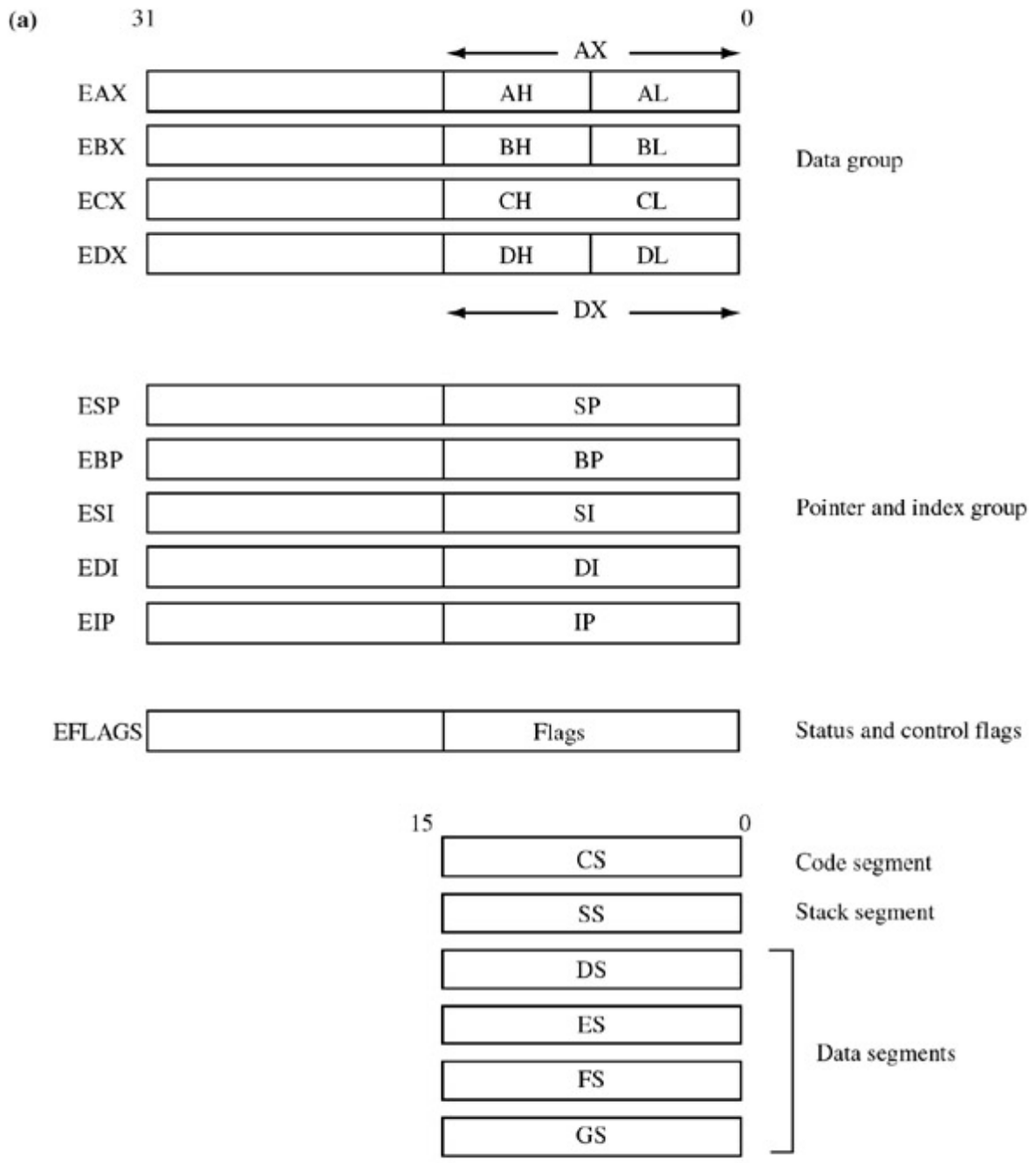
80386 Programming Model: GPR

- General Purpose Registers (Ia)
  - Data & Addresses Groups
    - Data/Pointer/Index registers
    - 32-bit width (max)
  - Data Group:
    - 8-bit: AL, AH, BL, BH, CL, CH, DL, DH
    - 16-bit: AX, BX, CX, DX
    - 32-bit: EAX, EBX, ECX, EDX
  - Accumulators:
    - 8-bit: AL, 16-bit: AX, 32-bit: EAX

80386 Programming Model: GPR

- General Purpose Registers (Ib)
  - Data & Addresses Groups
    - Data/Pointer/Index registers
    - 32-bit width (max)
  - Pointer & Index Group:
    - keep offset (logic address) relative to base address of a segment
    - 16-bit: SP, BP, SI, DI, IP =>
    - 32-bit: ESP, EBP, ESI, EDI, EIP

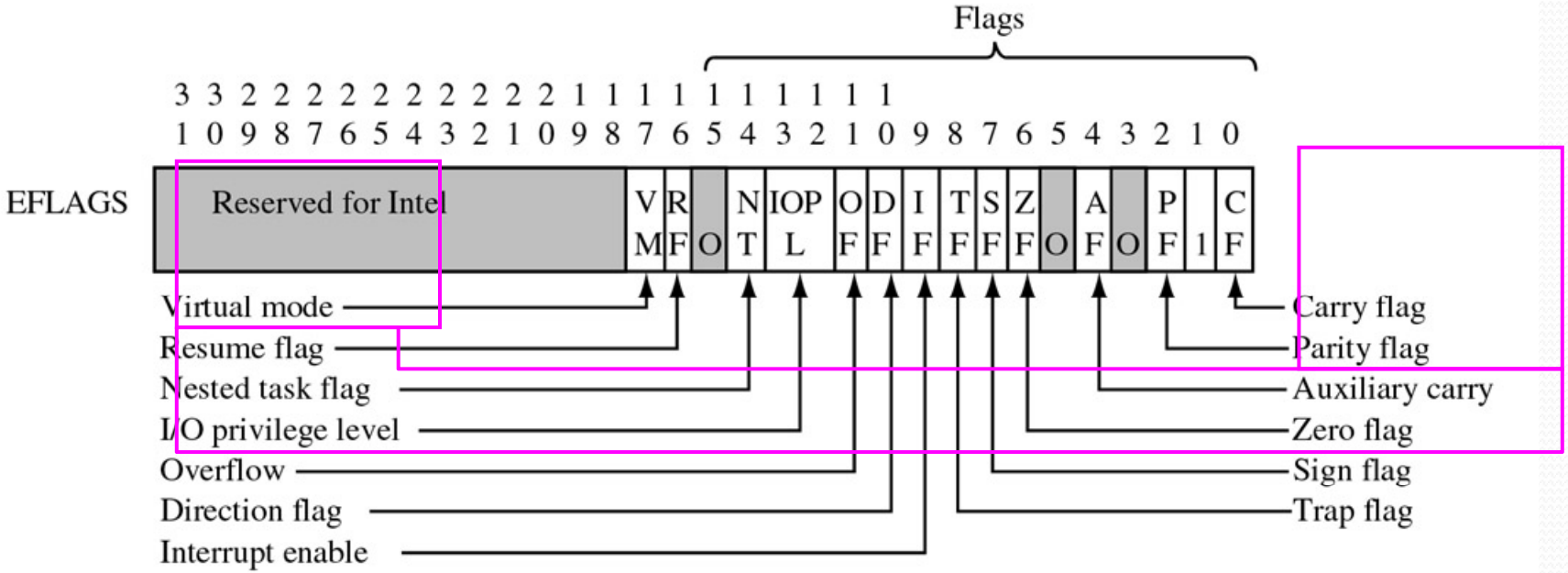
FIGURE 3-12.a
Programming model for the 80386. The general-purpose registers (a) are used by applications programmers. The special-purpose registers (b) are intended to be used by the operating system software.



80386 Programming Model: GPR

- General Purpose Registers (II)
  - Status & Control Flags:
    - EFLAGS: 32-bit, 4 new flags
    - VM: used to switch to V8086 mode
    - RF: resume from debug mode to normal execution
      - (used with debugging registers)
    - NT: nested task (current task was called from another task)
      - To determine type of return instruction
    - IOPL: current I/O privilege level (2-bit, PL 0-3) required to execute I/O instructions
      - OS control over I/O access

FIGURE 3-13 The 80386 flag word is 32 bits long. Four new flags have been added compared to the 8086: VM, RF, NT, and IOPL. (Courtesy of Intel Corporation.)



Note: O indicates Intel reserved; Do not define.

80386 Programming Model: GPR

- General Purpose Registers (III)
  - Segment Group:
    - CS, SS, DS, ES + FS, GS (new, not as default segment)
    - Remain 16-bit (NOT 32-bit)
    - Real mode: as segment base (for 8086 mode operation)
    - Protected mode: pointer to description table
      - NOT as base address of segment
      - base address is saved in a descriptor table

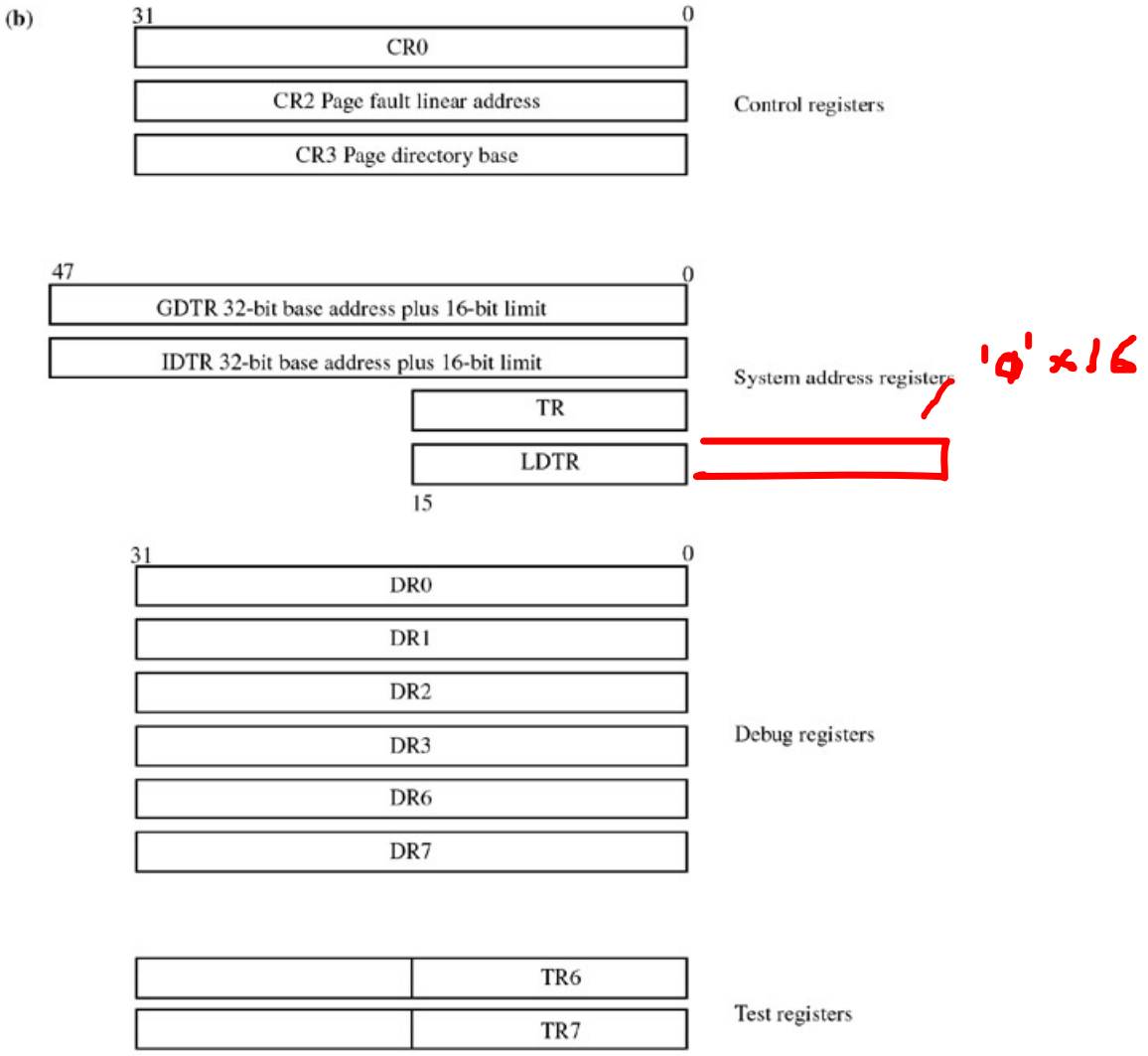
80386 Programming Model: SPR

- Special Purpose Registers (I) (3.11(b))
  - For Protected Mode control & testing
  - CR0 (32): used to enable paging mechanism, monitor task switching, enable co-processor emulation, select protected mode
  - CR2 (32) (page fault linear address): a reference to a page/segment that must be loaded into memory
  - CR3 (32) (page directory base): base address of the page table
    - Table: starting address of each page frame and access information of that frame

80386 Programming Model: SPR

- Special Purpose Registers (II)
  - System address registers x 4
    - GDTR/32(+limit/16), IDTR/32(+limit/16), TR/16, LDTR/16(+16 0's)
    - Descriptor table information (more ...)
  - Debug registers (32) x 6
    - To set program break points
  - Test registers (16) x 2
    - To test the RAM in Translation Lookaside Buffer (TLB, for virtual-to-physical address translation) (more ...)

FIGURE 3-12.b Programming model for the 80386. The general-purpose registers (a) are used by applications programmers. The special-purpose registers (b) are intended to be used by the operating system software.



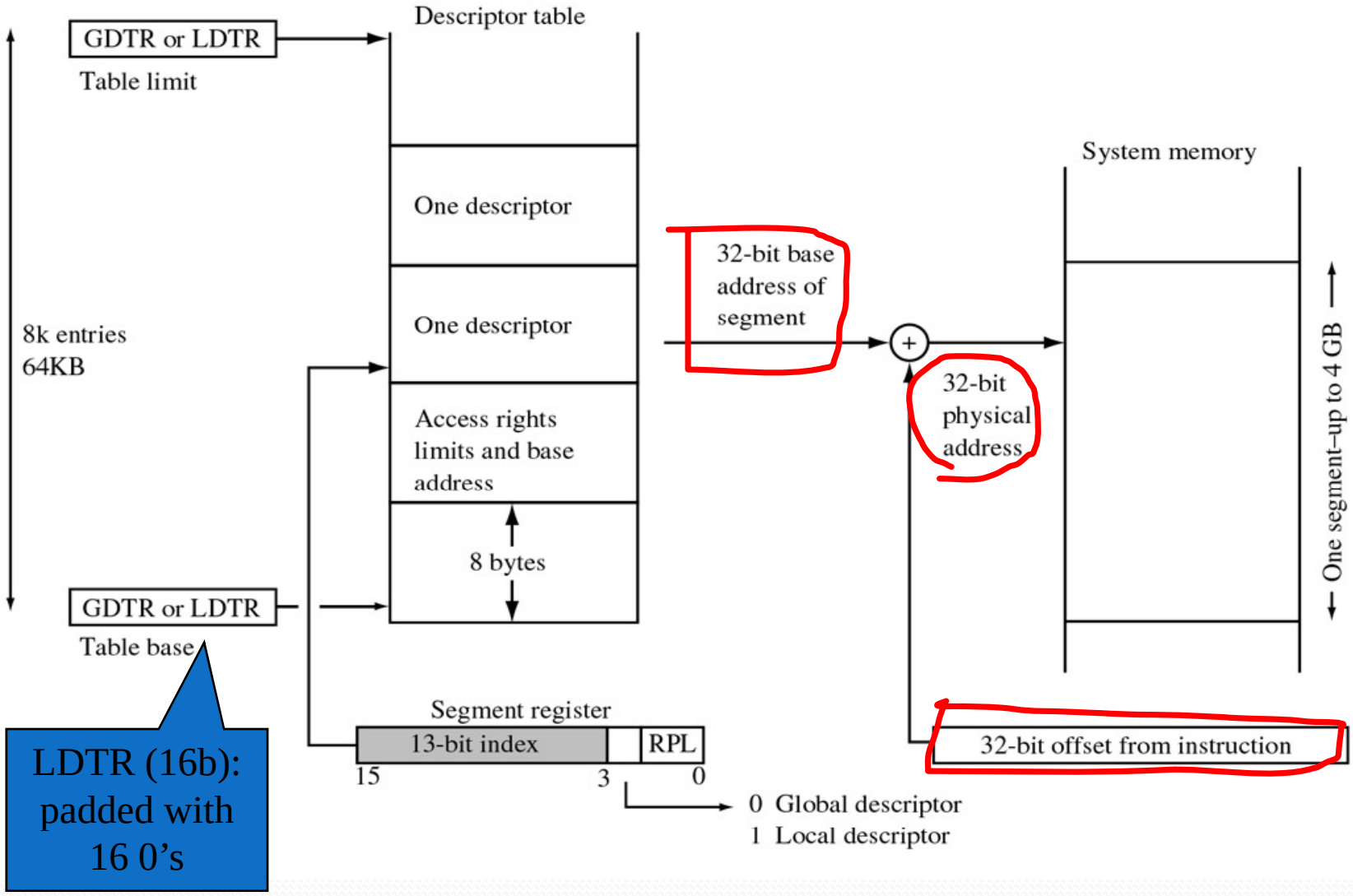
80386 Programming Model: Segments

- Memory Management
  - Segment descriptors
    - Function: keep base address, size, access rights
    - 3 types of tables: global (GDT), local (LDT), interrupt (IDT)
    - GDT: pointing to segments that may be accessible to *all tasks*
    - LDT: pointing to segments associated with *a given task*
      - Each task may have one LDT
    - IDT: point to the starting addresses of *interrupt service routines*
      - Max. 256 ISR's (processor faults, hardware/software INT's)

80386 Programming Model: Segments

- Memory Management (cont.)
  - Segment addressing:
    - Segment Register = Index + 1-bit Global/Local Flag + RPL (requesting privilege level)
    - Index => point to a descriptor table
    - Address = base (in table) + offset (from instruction)
  - 13-bit index: 8K descriptors
    - (8K GDT+8K LDT)x4G = 64T virtual space
    - =(13+1+32)= 46-bit virtual address (1: GDT/LDT)
  - Base+limit is stored in GDTR/LDTR
    - LDTR: 16-bit register, padded with 16 0's (i.e, 64K descriptor table boundaries) to form a 32-bit base address to LDT

FIGURE 3-15 80386 Protected Mode addressing. Physical addresses are computed by adding the instruction offset to the segment base address stored in a descriptor table. The upper 13 bits of the segment register are used to point to a specific descriptor. The base address and limit of the descriptor tables are stored in the global and local descriptor table registers (GDTR and LDTR).



Protected Mode Architecture (cont'd)

- Segment descriptor provides attributes of a segment
  - » 32-bit base address
  - » 20-bit segment size (bytes/pages)
  - » Control and status information

E (xecutable)
C/ED(confirming/executable direction)
R/W (read/write)
A(accessed)

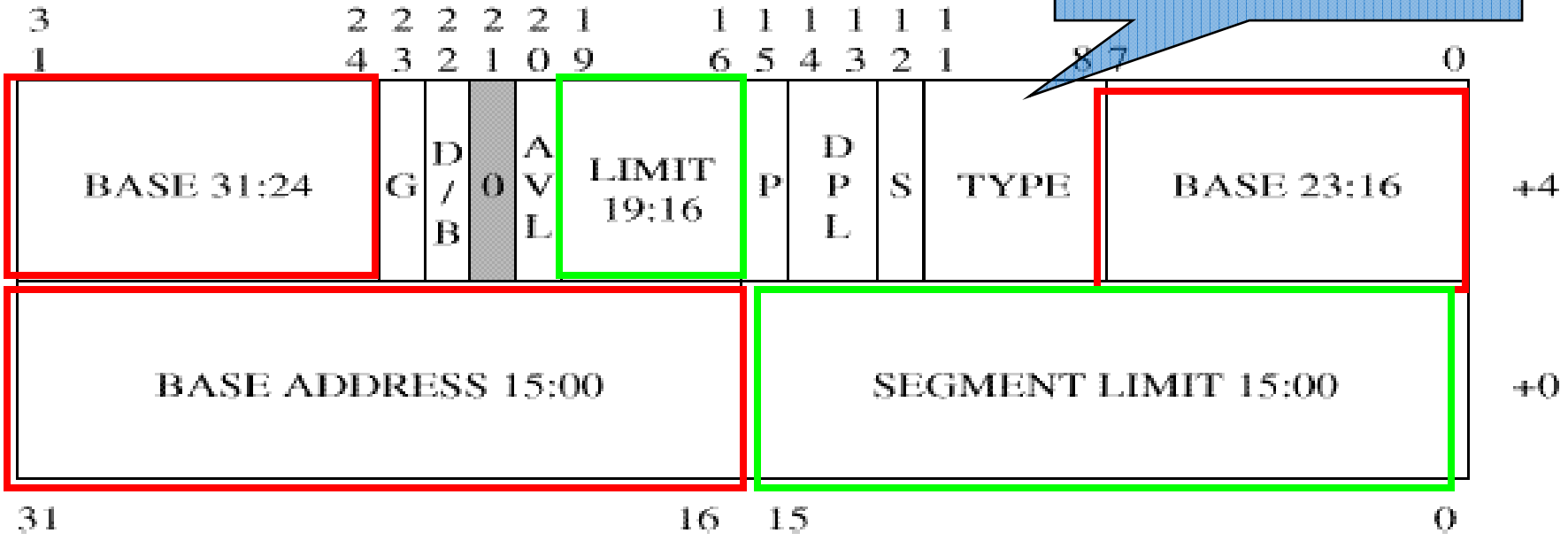


FIGURE 3-14 In Protected Mode each segment register points to the base of a descriptor table. Entries in these tables, called descriptors, are eight bytes long and specify the starting address of the segment, its size limit, and its attributes.

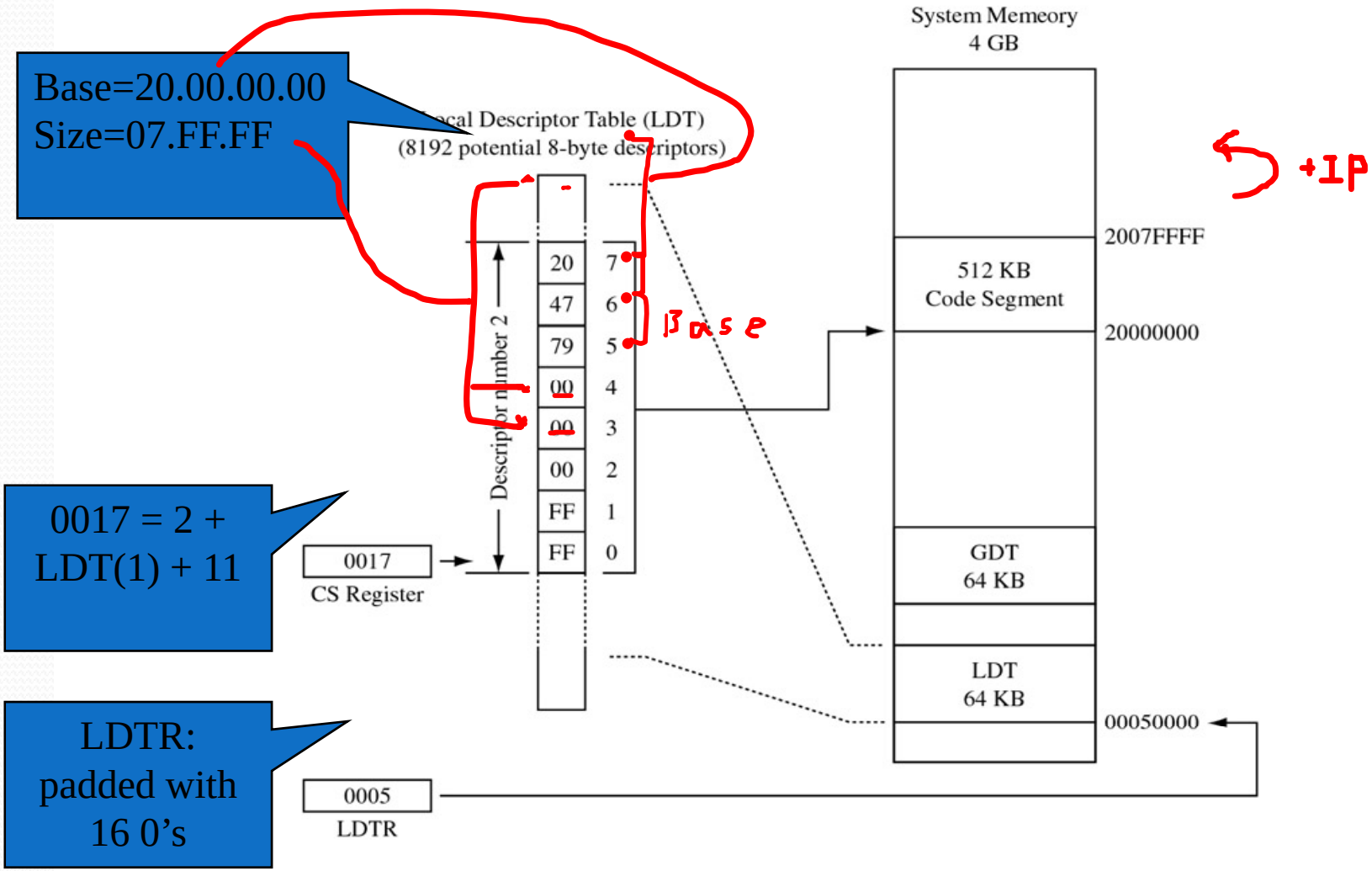
Limit/Size:
G=0 ⇔2^20 bytes (=1MB)
G=1⇔2^20 pages (4k)
(= 4GB)

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	
Segment Base Address B24–B31								Byte 7
G	D	O	AVL	Segment Size Limit L16–L19				Byte 6
P	DPL		S	E	C/ED	R/W	A	Byte 5
Segment Base Address B16–B23								Byte 4
Segment Base Address B8–B15								Byte 3
Segment Base Address B0–B7								Byte 2
Segment Size Limit L8–L15								Byte 1
Segment Size Limit L0–L7								Byte 0

Notes:

Segment Base Address	32-bit physical starting address of the segment.
Segment Size Limit	The segment is specified in bytes with a 1 MB maximum (G = 0) or 4 KB pages with a 4 GB maximum (G = 1).
P (present bit)	The segment is present in main memory when this bit is set. If P = 0 the operating system must bring the segment into memory from the hard disk.
DPL (descriptor privilege level)	The privilege level can vary from 00 (most privileged) to 11 (least privileged).
S (segment descriptor)	If S = 0 the segment is a system descriptor. If S = 1 the segment is a code or data segment.
E (executable)	The segment is executable (E = 1) or data (E = 0).
C/ED (conforming/executable direction)	If E = 1: Then if C/ED = 1 code segment can be executed only when CPL ≥ DPL otherwise these bits are ignored. If E = 0: Then if C/ED = 1 expand up the segment otherwise if C/ED = 0 expand down the segment.
RW (read/write)	Access is read-only (RW = 0) or read/write (RW = 1)
A (accessed)	If the segment has been accessed A = 1. This bit allows the operating system to decide if a segment needs to be saved or can be overwritten (trashed).
G (granularity)	This bit sets the size of the segment (see Segment Size Limit).
D (default operation size)	When D = 0 16-bit addressing and CPU registers are assumed. When D = 1 32-bit addressing and CPU registers are assumed.
AVL	This bit is available to the operating system.

FIGURE 3-16 Example showing register LDTR pointing to the base of the local descriptor table (LDT) at address 00050000H. The CS register is pointing to descriptor number two in this table. The eight bytes that make up this descriptor specify a 512 KB memory segment beginning at address 20000000H.



80386 Programming Model: Paging

- Memory Management (cont.)
  - Paging: (CRo bit31=enable paging)
    - 13-bit index + Global/Local flag = 8K x 2 descriptors
    - Each descriptor point to 4G (2**32)
    - Addressing space for a task: 16K x 4 G = 64T
    - Need a paging mechanism to support virtual memory if less than 64T physical memory (only 4G for 386)
  - A page translation mechanism is added when paging is enabled (Fig. 3.14)
    - To Compute physical address within a 4 K page frame & the address of the page frame
    - via. Page Directory & Page Table

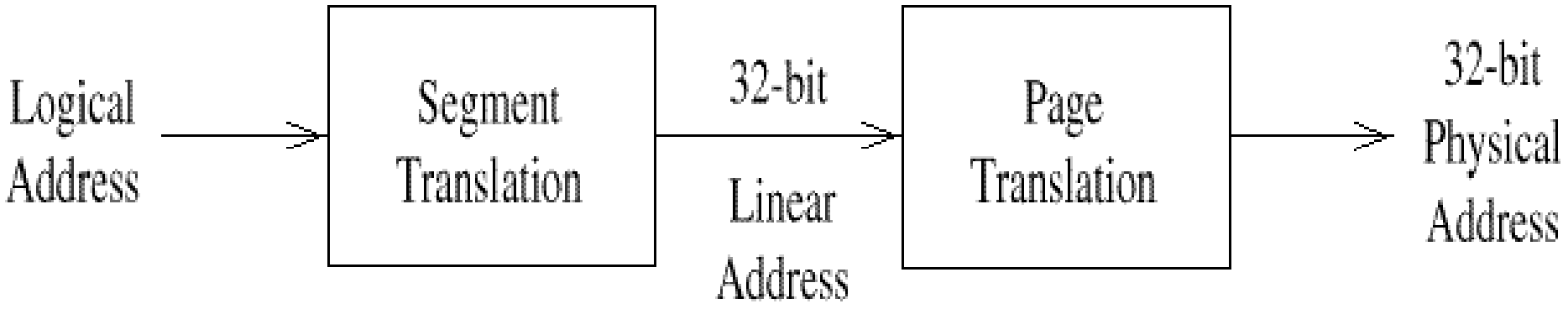
80386 Programming Model: Paging

- Memory Management (cont.)
  - Page Fault: requested page is not in real memory
  - Page Swapping:
    - Swap out unused and swap in requested pages
    - normally by LRU (Least Recently Used) strategy
  - TLB (Translation Lookaside Buffer):
    - Contains the addresses of the 32 most recently accessed page frames (coverage: 4K x 32 = 128K bytes)
    - For fast page look-up
    - For reducing page miss: 98% page hit (in TLB)

Protected Mode Architecture

- In protected mode, Pentium supports
  - * More sophisticated segmentation
    - » Segmentation can be made invisible (flat model)
  - * Paging for virtual memory
    - » Paging can be turned off

Dir(10)+Pag(10)+Offset(12) ⇔ 4k repageable pages ⇔ [VM image]



Memory Paging

- The **memory paging mechanism** allows any physical memory location to be assigned to any linear address.
  - **Linear address:** the address generated by a program.
  - **Physical address:** the actual memory location accessed by a program.
- With memory paging, the linear address is invisibly translated to any physical address.
  - Data are not stored where the program think of

Figure 2–13 The paging mechanism in the 80386 through Core2 microprocessors.

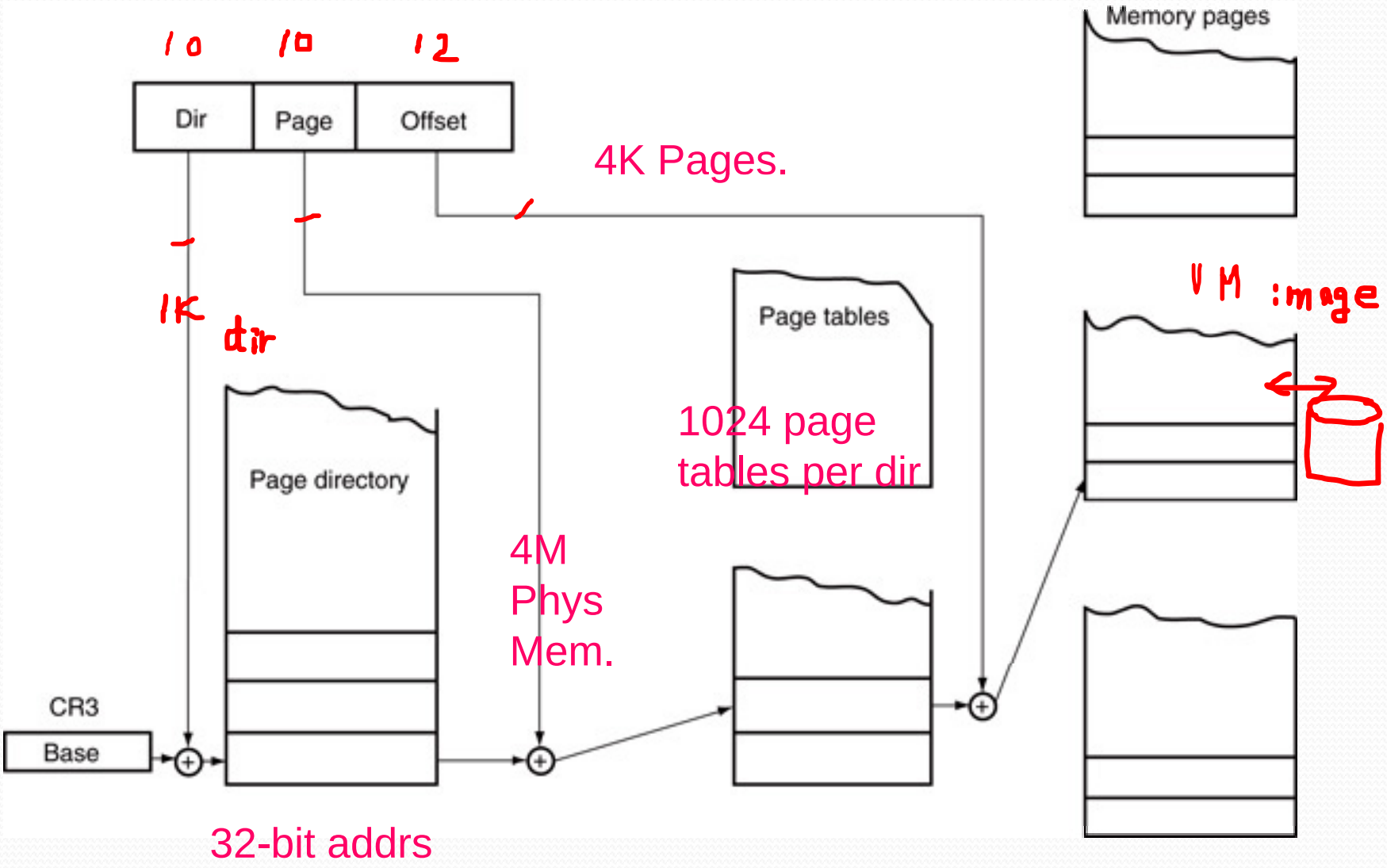


Figure 2–11 The control register structure of the microprocessor.

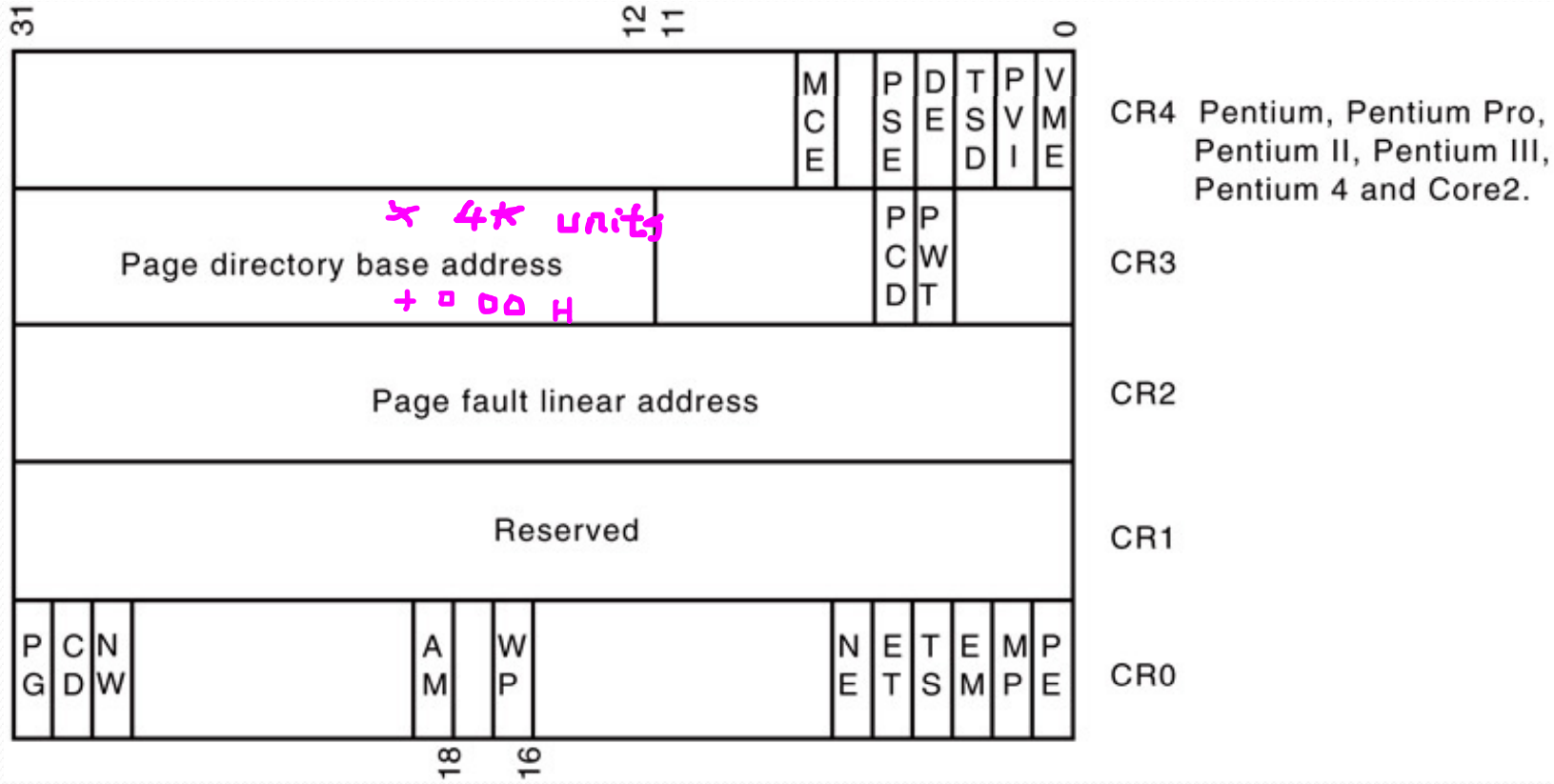


Figure 2–12 The format for the linear address (a) and a page directory or page table entry (b).

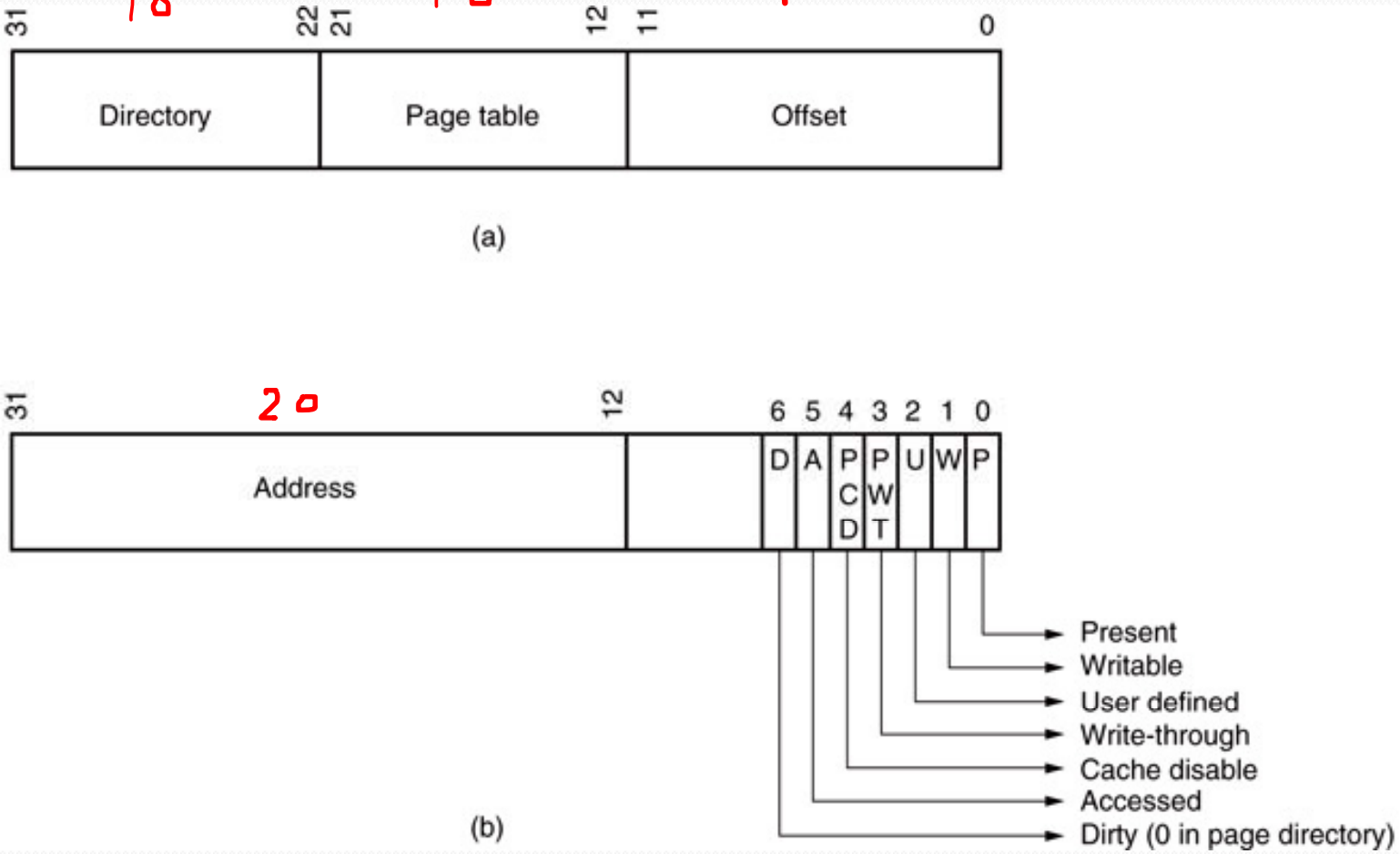


FIGURE 3-17 When paging is enabled, linear addresses are translated into physical addresses via the Page Directory and Page Translation tables.

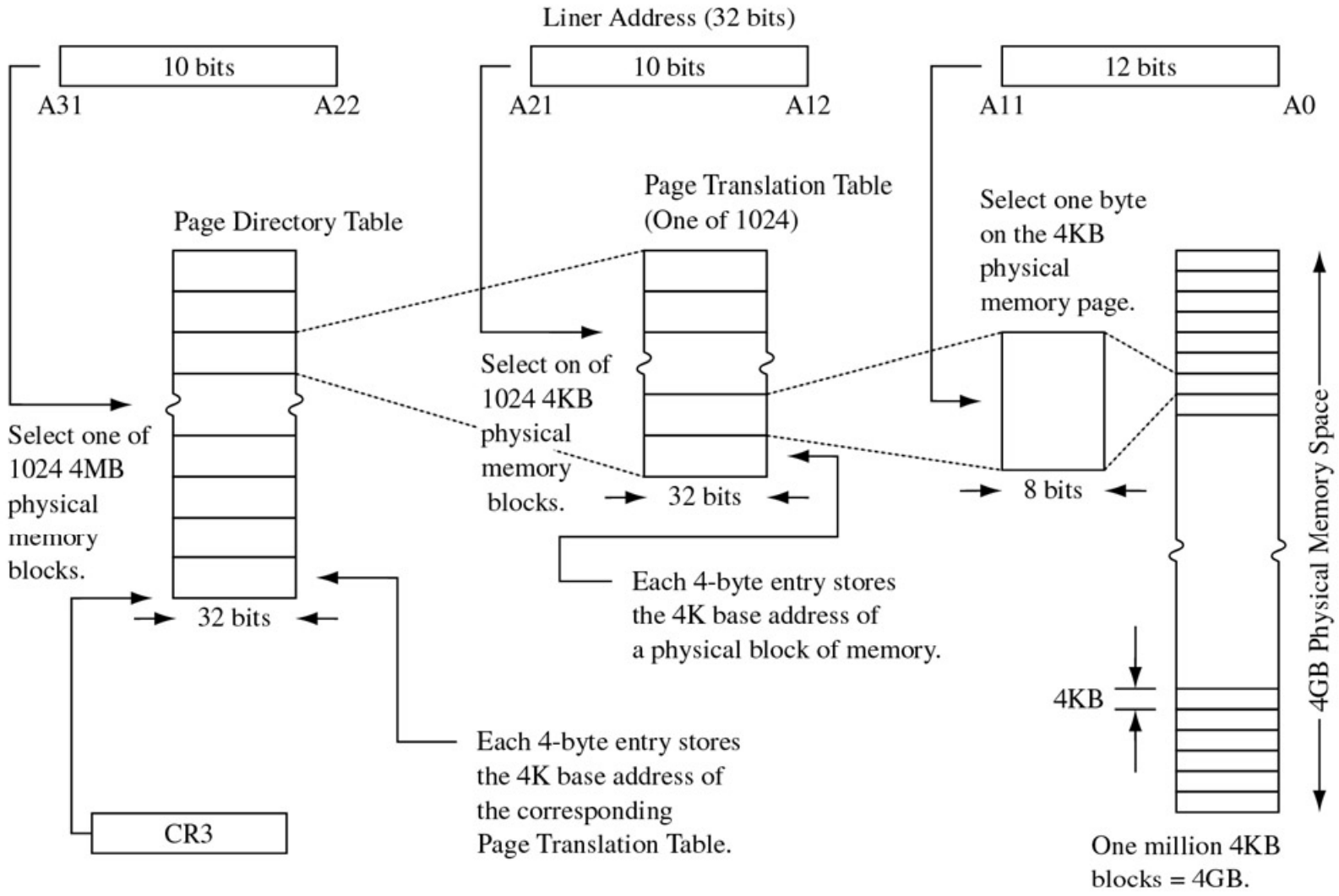
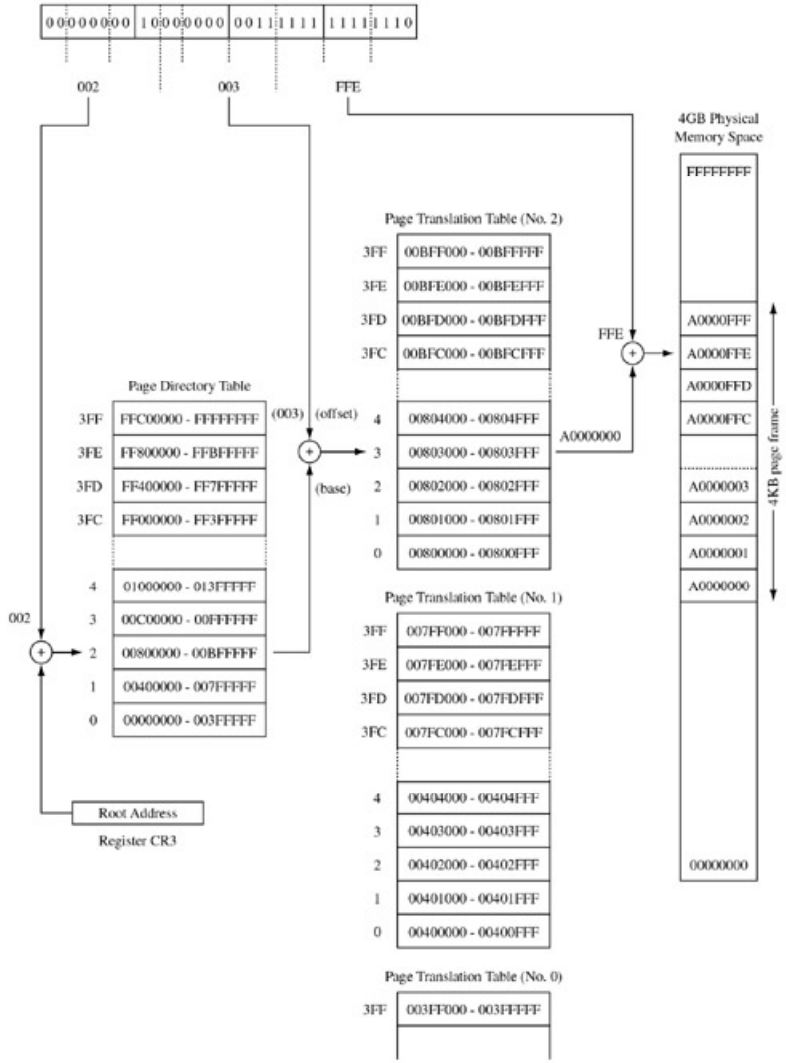


FIGURE 3-18 Example showing how linear address 00803FFE is translated into physical address A0000FFE.



80386 Programming Model: PL

- Protection:
  - assign PL (Privilege Level) to resources to prevent lower privilege tasks from accessing high privilege resources
  - PL 0~3, 0: highest privilege
  - task: CPL
  - Instruction/segment register: RPL
  - data segment: DPL
  - Rule: EPL > DPL => general protection fault
    - where EPL=max(RPL,CPL)

80386 Programming Model: PL

- Protection (cont.): Gates
  - Special descriptors that allows access to higher PL tasks from lower PL tasks
    - By accessing lower PL gates, which can access higher PL resources: EPL <= DPL(gate)
- Types
  - Call gates: provide access to high PL codes
  - Task gates: for task switching
  - Interrupt gates: to specify ISR's
  - Trap gates: to specify trap (error) handling routines

80486 Model …

- Factors that make it faster than 386
- Operation of Direct mapped cache
- 486 cache diagrams
  - Cache RAM, Tag RAM, LRU SRAM
  - Data & control registers within FPU

80486DX

- 1989: a polished 386, 6 new OS level instructions
- virtually identical to 386 in terms of compatibility
- RISC design concepts
  - fewer clock cycles per operation, a single clock cycle for most frequently used instructions
  - Max 50MHz
  - 5 stage execution pipeline
    - Portions of 5 instructions execute at once

80486DX

- Highly Integrated:
  - On board 8K memory cache
  - FPP (equivalent to external 80387 co-processor)
- Twice as fast as 386 at any given clock rate
  - 20Mhz 486 ~= 40Mhz 386

80486SX

- 80486SX
  - NOT a 16-bit version for transition purpose
  - no coprocessor
  - No internal cache
  - For low-end applications
  - Max. 33Mhz only

80486DX2/DX4: Overdrive Chips

- Processor speed increased too fast
  - Redesign of microcomputer for compatibility becomes harder
  - Solution: Separating internal speed with external speed, improve performance independently
- 80486DX2/DX4 – internal clock twice/three times (NOT four times) the external clock: runs faster internally

80486DX2/DX4: Overdrive Chips

- System board design is independent of processor upgrade (less expensive components are allowed)
- Processor operate at maximum speed data rate internally
  - Only slow access to external data operates at system board rate
  - Internal cache offset the speed gap
- 486DX2 66: 66 internal, 33 external
- 486DX4 100: 100 internal, 33 external (3x)
- *Overdrive* sockets: for upgrading 486dx/sx to 486dx2/dx4 (with overdrive socket pin-outs)

486 Processor Features

- 386 features:
  - Real/Protected Modes
  - Memory Management
  - PL's
  - registers & bus sizes
- New features
  - 6 OS instructions
  - 8K/16K onboard cache (was external before 386)
  - FPU (was external)

486 Processor Features

- A Better 386
  - 5 stage instruction pipeline
    - IF/ID/EX => PF/D1/D2/EX/WB
    - PF: pre-fetch instructions => Q (2*16-bytes)
    - D1: determine opcode
    - D2: determine memory address of operands
    - EX: execute indicated OP
    - WB: update internal register

486 Processor Features

- Reduced Instruction Cycle Times
  - 5 stage instruction pipeline (e.g., Fig. 3.18)
  - instruction cycle times:
    - 8086: 4 CLK
    - 80386: 2 CLK
    - 80486: 1 CLK (⇔close to RISC)
    - about 2X faster than 386

486 Processor Model: 386+FPU+Cache

- 386 units retained: BIU, CPU, MMU
- new: FPU (80387) + Cache (8K/16K)
- FPU:
  - 387 onboard
    - 0.8 u => #transistors increased (275K => 1+ millions)
    - simplified system board design
    - speedup FP operations

FIGURE 3-11 The processor model for the 80386 microprocessor consists of the bus interface unit (BIU), central processing unit (CPU), and the memory management unit (MMU).

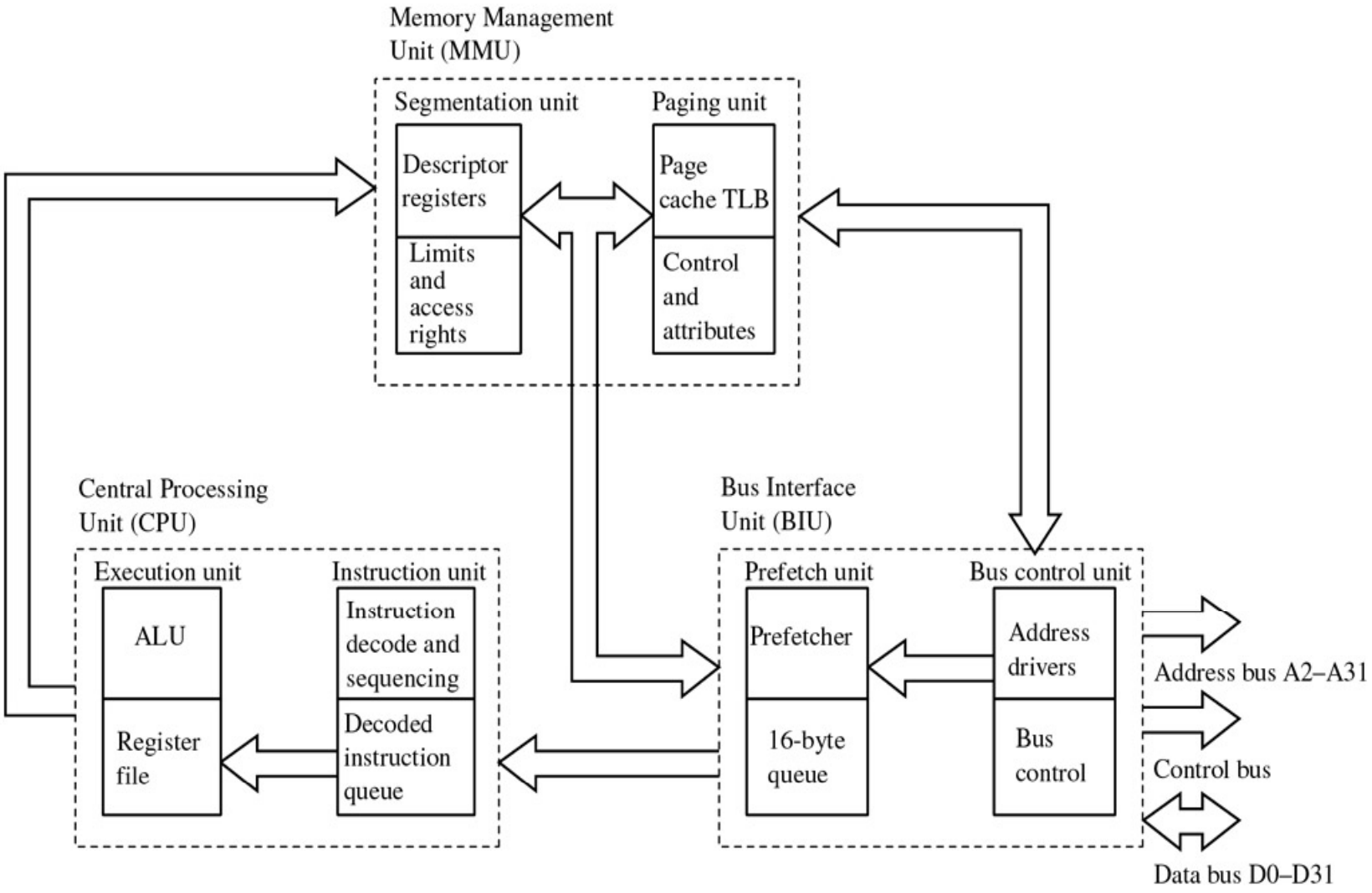


FIGURE 3-11 The processor model for the 80386 microprocessor consists of the bus interface unit (BIU), central processing unit (CPU), and the memory management unit (MMU).

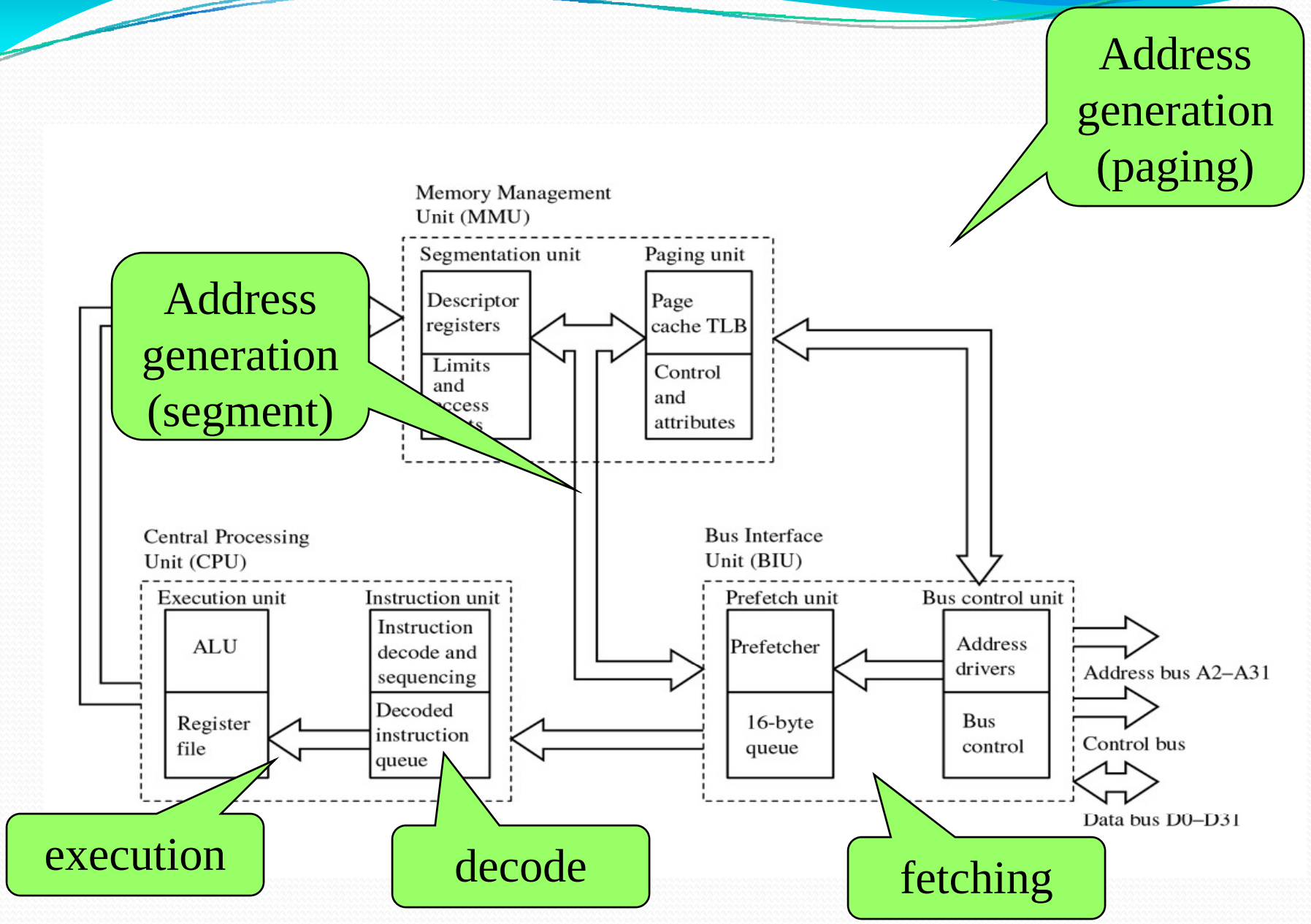
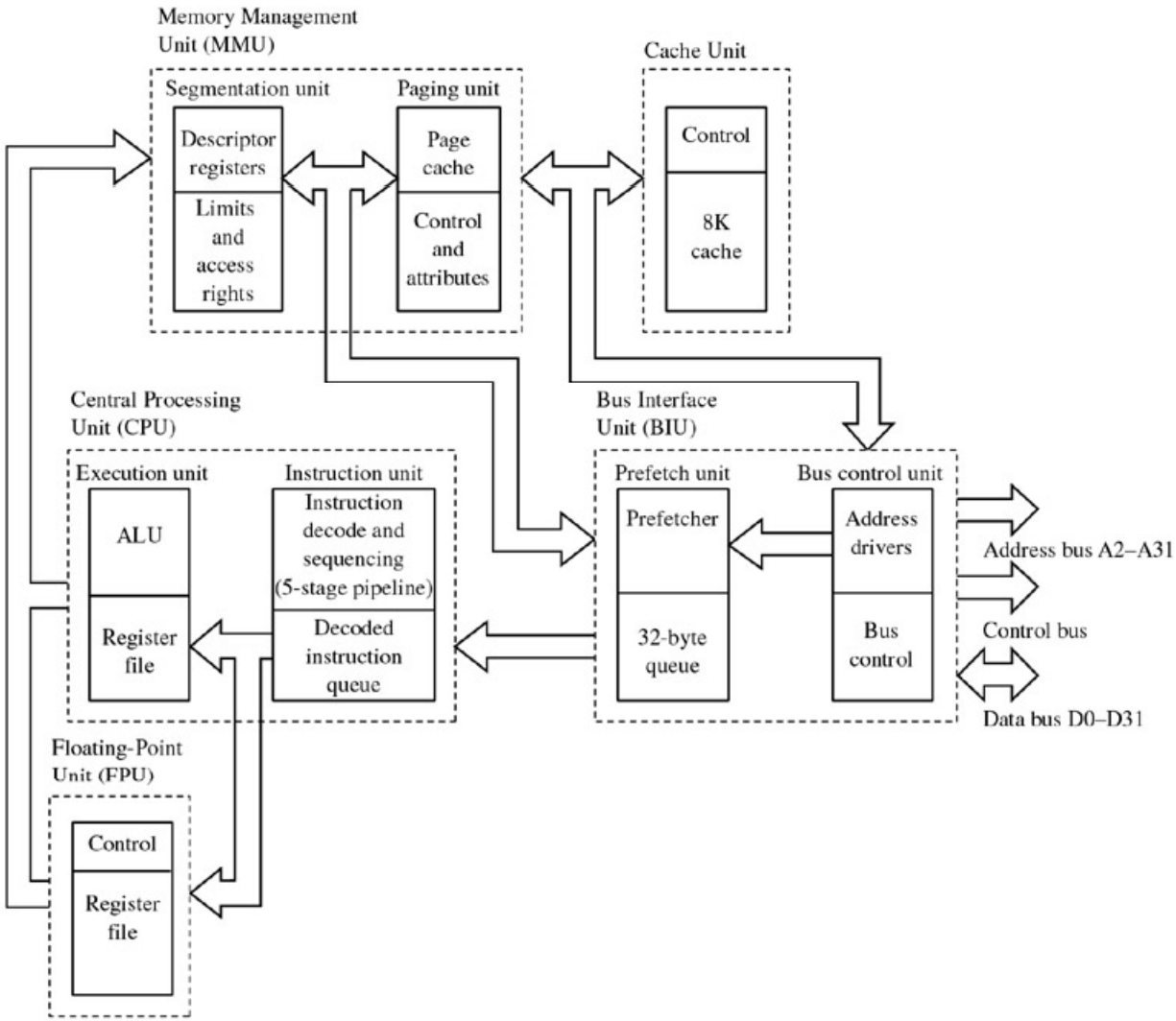
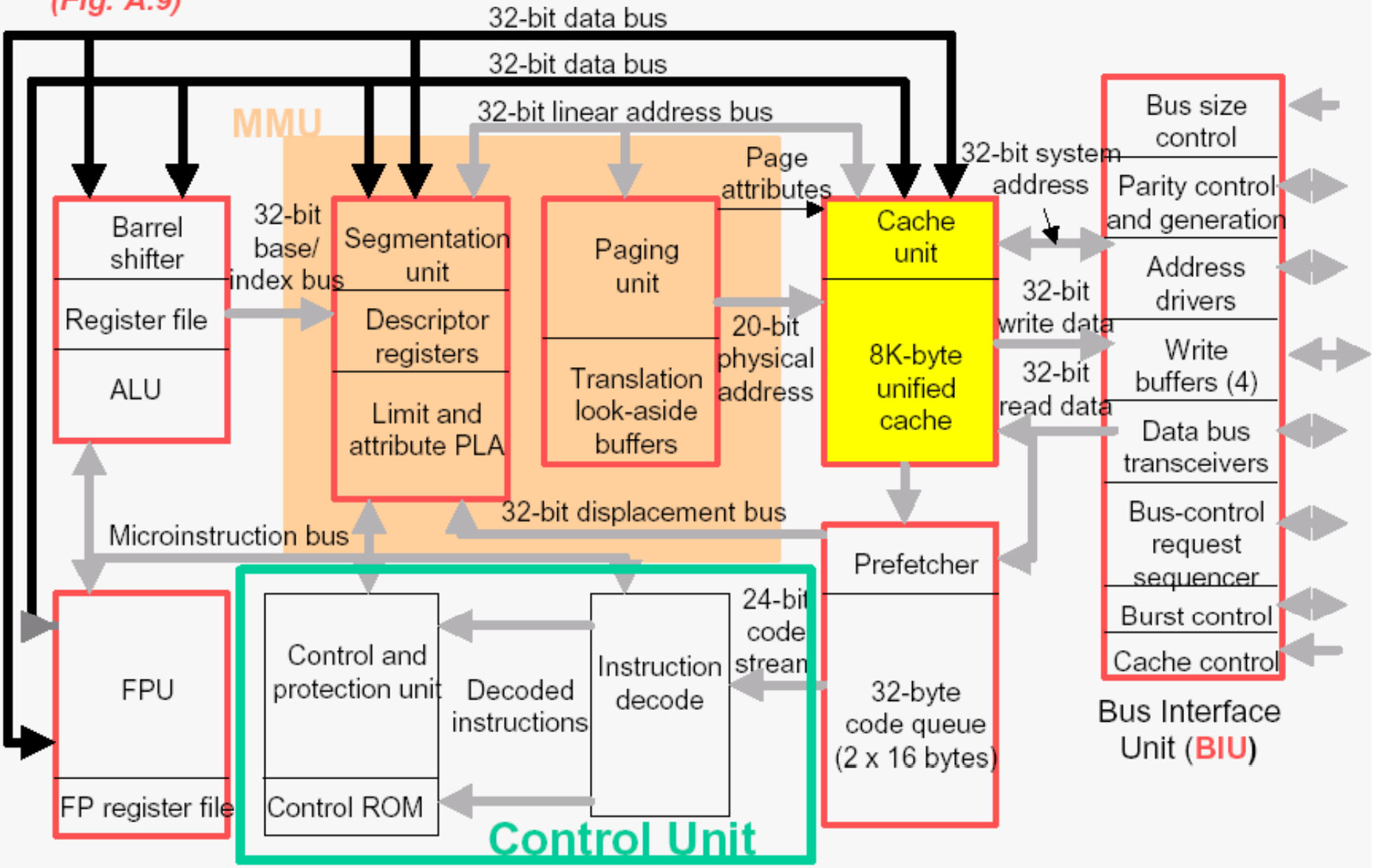


FIGURE 3-24 The processor model for the 80486 microprocessor is the same as that for the 80386 except for the on-board cache and floating-point cache.



An internal unified cache (Intel 486)

(Fig. A.9)



486 Processor Model: Cache

- Cache (8K/16K (dx4))
  - Function: bridge processor memory bandwidth
    - 8088: 4.77MHz
    - 80486: 50MHz
    - Pentium: 100MHz
    - Pentium Pro: 133 MHz
    - ⇔ Main Memory (DRAM): relatively slow
  - Fast Static RAMs (SRAM) as cache
    - Processor ⇔ Fast Cache (lines) ⇔ Slow Main Memory (blocks)
    - A running block of main memory is copied to cache line when needed and not in cache
      - Cache miss: An unused cache line is updated and trashed if all cache lines are being used while trying to copy

486 Processor Model: Cache

- Organization:
  - Size: 8K = 2K x 4
  - Mapping: 4-way set associative
    - 4 direct mapped caches wired in parallel
    - each block maps to a set of 4 cache lines
  - Unified: data & code in the same cache
  - Write-through update policy: update cache and memory page on write operations

486 Processor Model: Cache

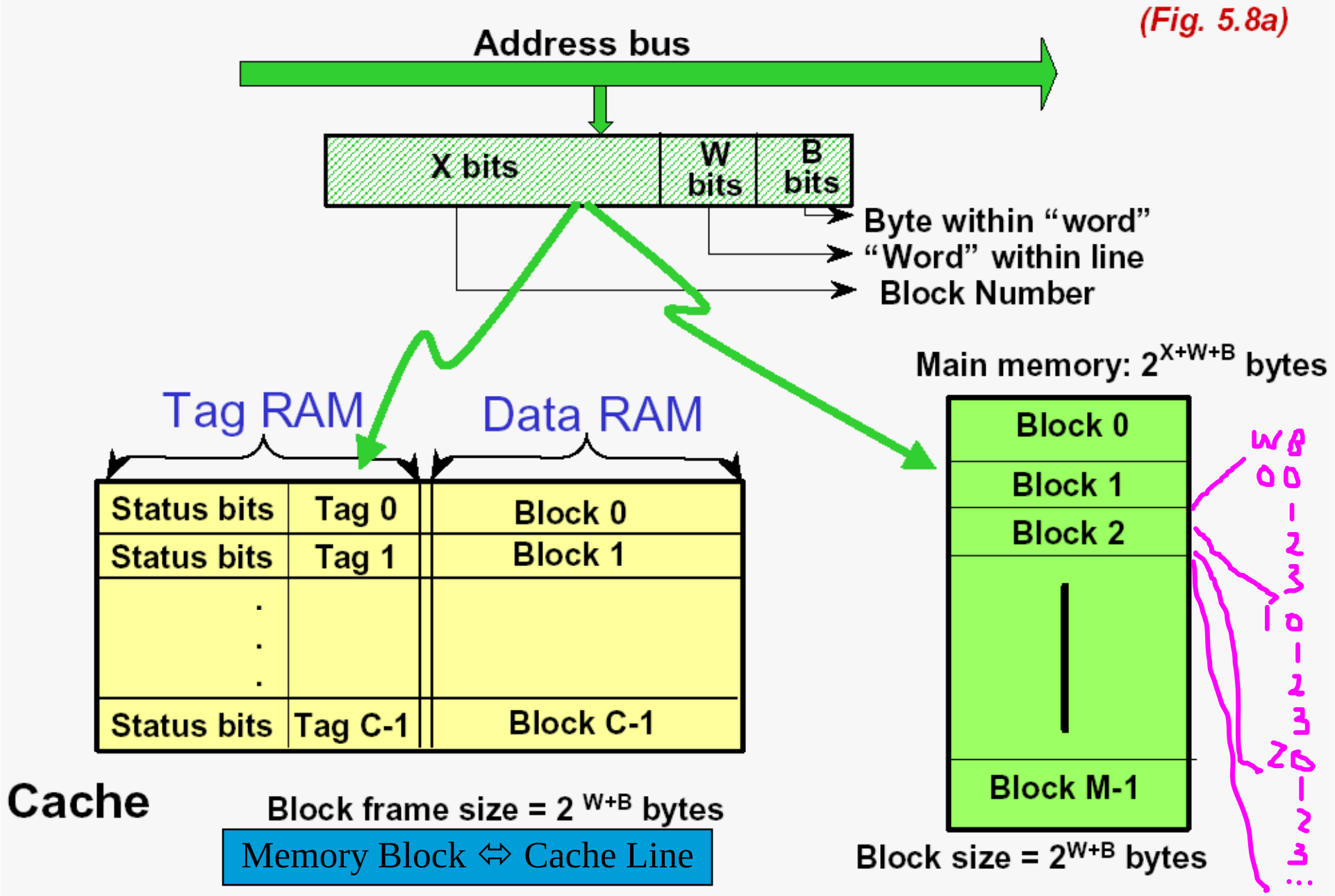
- Locality: Why caches help?
  - spatial locality: e.g., array of data
  - temporal: e.g., loops in codes
- Operations on cache hit/miss
  - Hit: memory copy is found in cache, use cached copy
  - Miss: memory copy not found in cache
    - Load memory copy to one of allowed free/unused cache blocks
    - If none is free, replace a less recently used (update replaced block)
- Size of cache line: 128-bit (16-byte) cache lines
  - 32-bit x N to catch locality (N=4)
  - 128-bit = 16-byte = 4 x 32-bit double-words

486 Processor Model: Cache

- Mapping:
  - Memory => Cache: many-to-many
    - Need to remember where the cached data came from
      - To decide if a memory copy is in cache
      - To update cache copy to right memory block (when replaced)
  - Cache = Data RAM + Tag RAM
    - Data RAM: save memory data
    - Tag RAM: save memory address & access status information

Address to cache and main memory

(Fig. 5.8a)

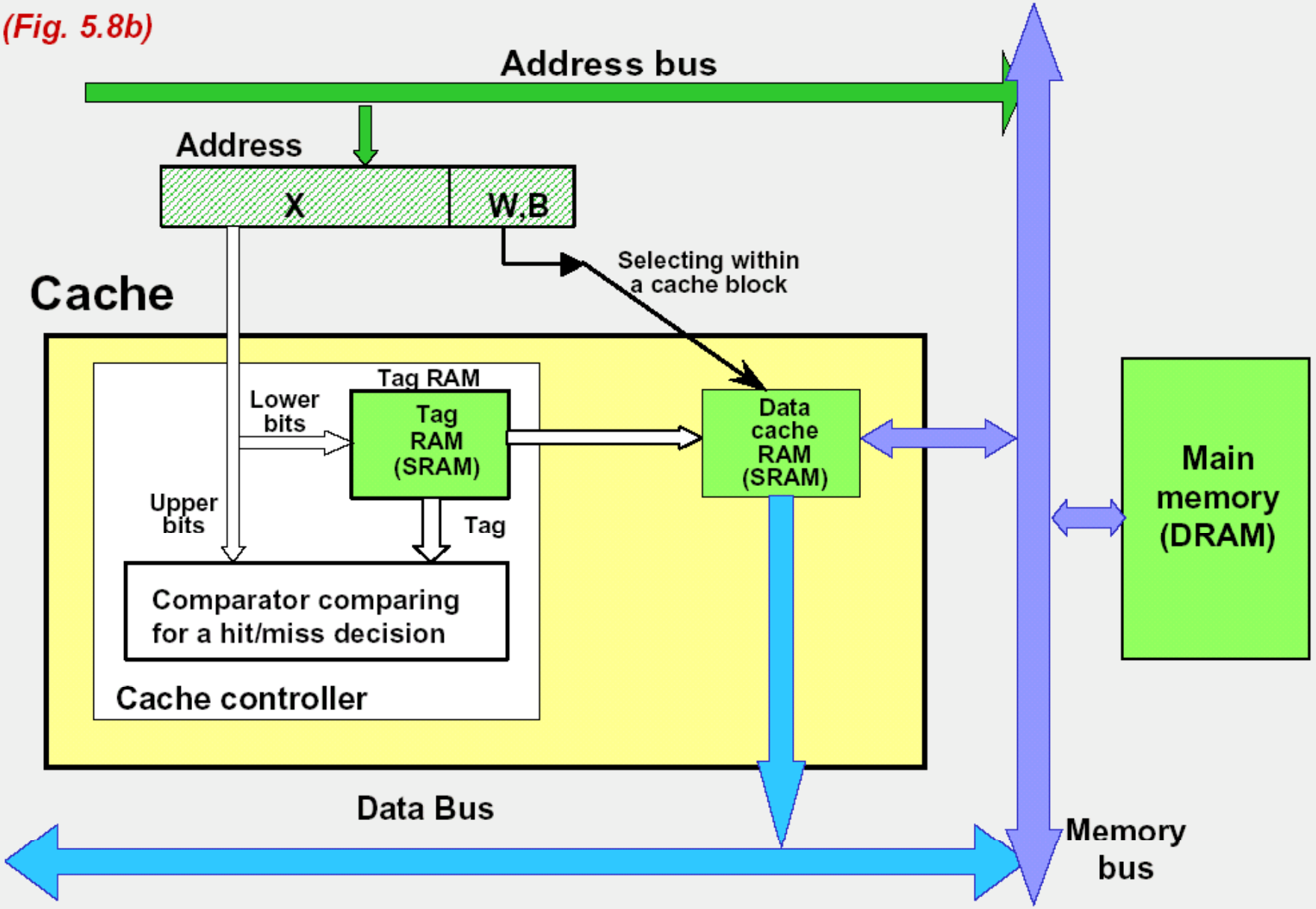


486 Processor Model: Cache

- main memory size = 2**(X+W+B) bytes
  - main memory divided into blocks of size 2**(W+B) bytes
  - main memory address: X+W+B bits
  - X (the “block number”) is treated differently depending on the cache organization
  - E.g., 16-byte as a block (⇔ a cache line)
    - B=2 [32-bit-word=4-byte=2**2]
    - W=2 [16-byte=2**4=2**2x2**2=2**2 x 32-bit-word]
    - X=32-4=28 [2*28 blocks, each having 16 bytes]

Block diagram of a cache

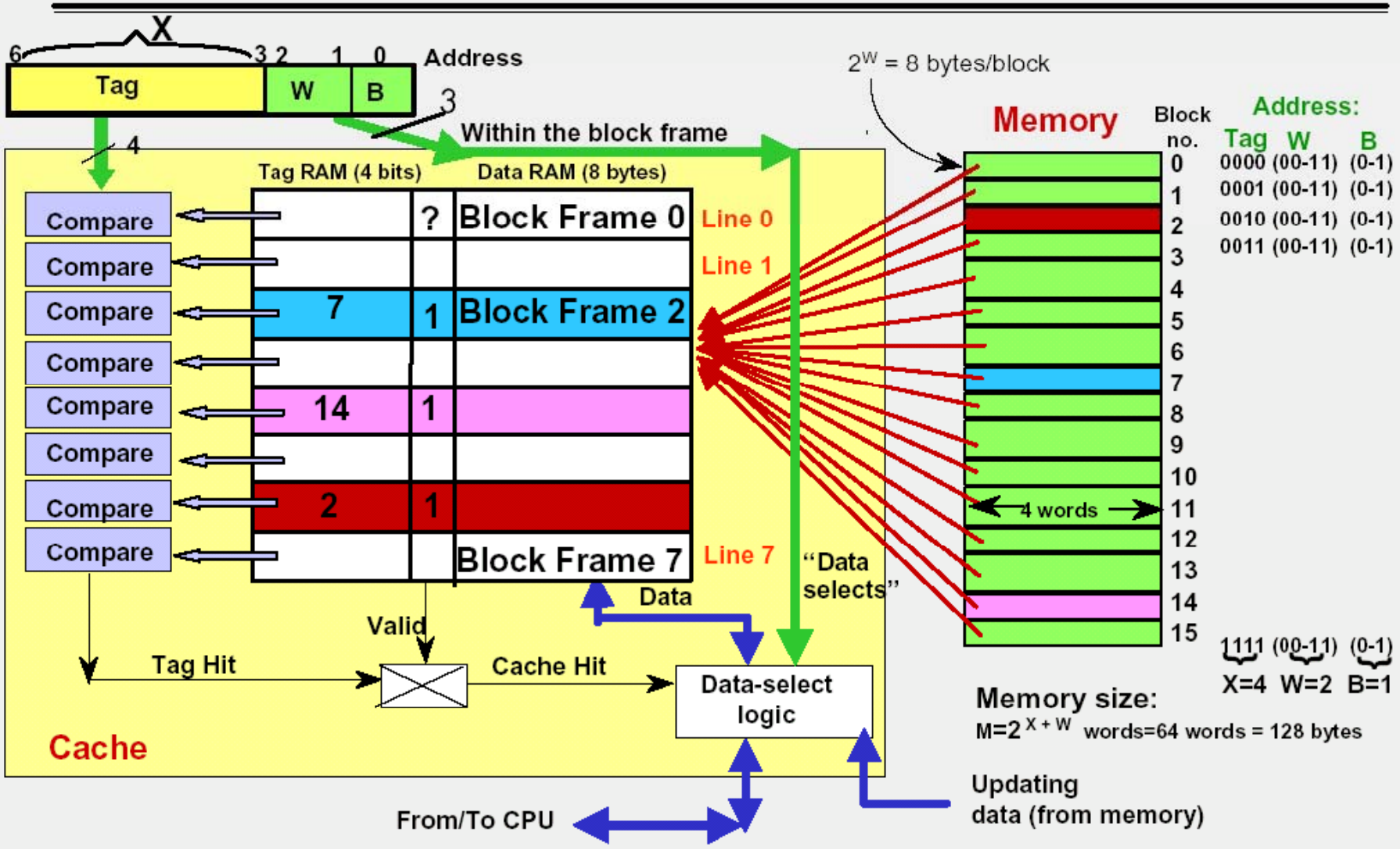
(Fig. 5.8b)



486 Processor Model: Cache

- 3 methods of mapping
  - Fully associative: map any memory block to any cache line
  - Direct map: map each memory block to specific cache line
  - Set associative (M-way): map each memory block to a set of M cache lines

Fully associative cache



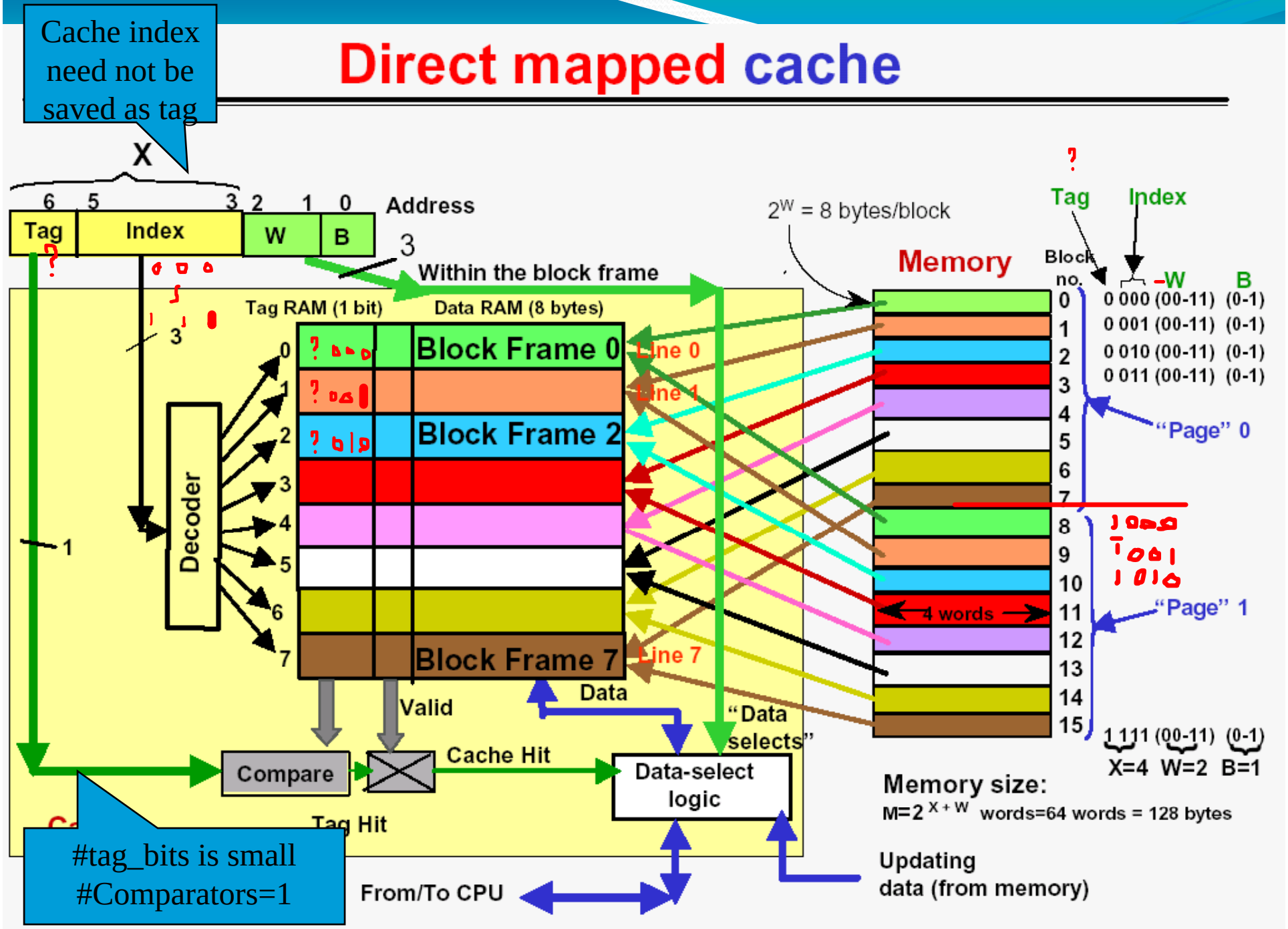
Fully associative cache mapping: any block from main memory can be put anywhere in the cache;
number of sets S = 1, therefore, no "set field" is required;
number of lines/set is K = C = 8

8-byte/block & 7-bit address…

Cache: Fully Associative Cache

- Fully associative: memory block to any cache line (有空位就停:停車容易,找車難)
  - Flexible to save memory blocks into cache lines
    - Small trashing rate
  - #tag_bits is largest
    - Since all X bits for block numbers are used as tags
  - #comparators=#cache_lines
    - parallel comparison with all cache lines
    - largest number of comparators

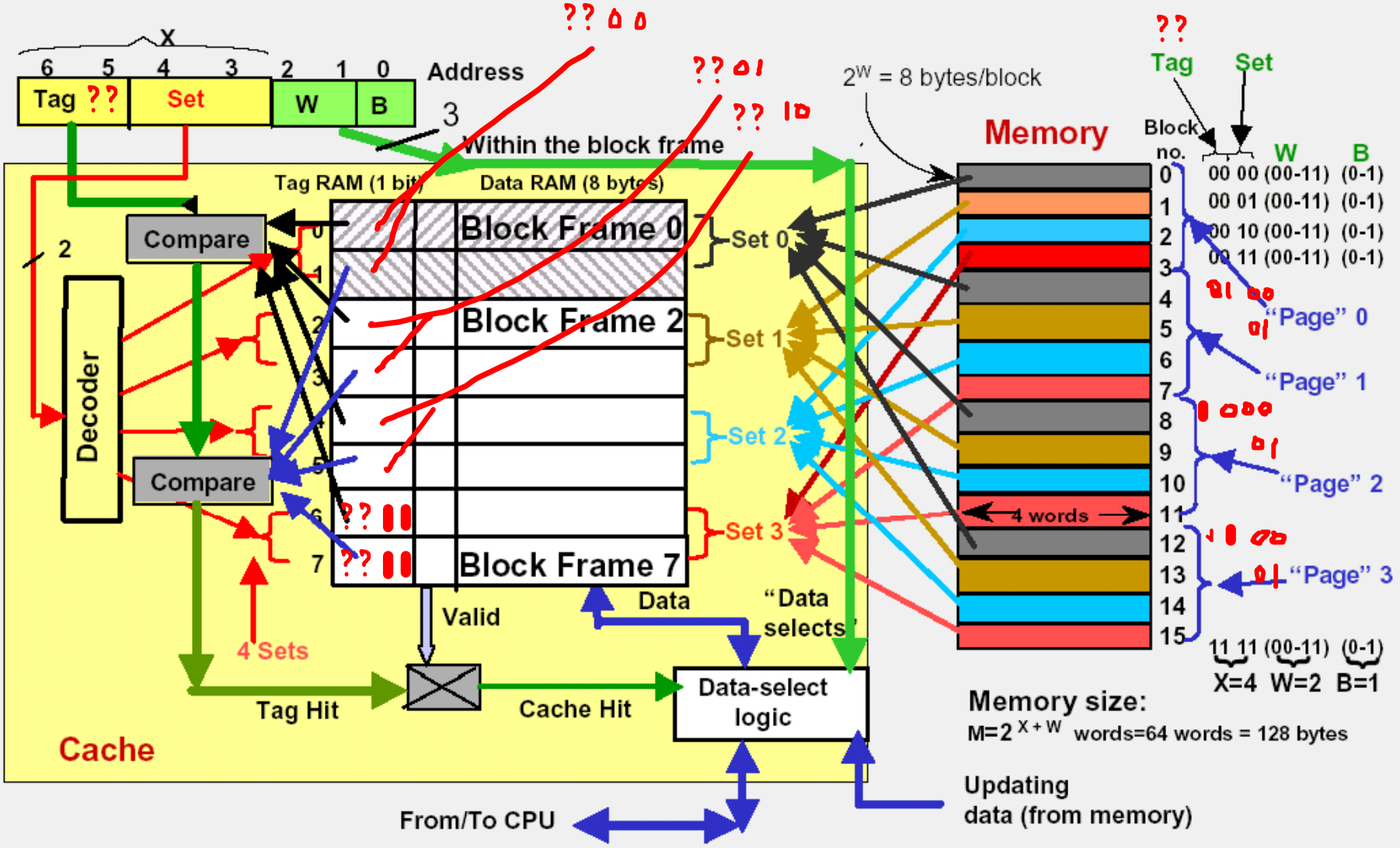
Direct mapped cache



Cache: Direct Mapped Cache

- Direct map: memory block to specific cache line (指定車位:停車難,找車容易)
  - Easily mapped to: block_num % number_of_cache_lines
  - #tag_bits: smallest
    - since index to cache line need not be saved as tags
  - #comparators=1 (the cache-index-selected one)
  - Trashing: repeatedly access memory that maps to the same cache; repeatedly swapped in/out; increase access time

2-way set associative cache



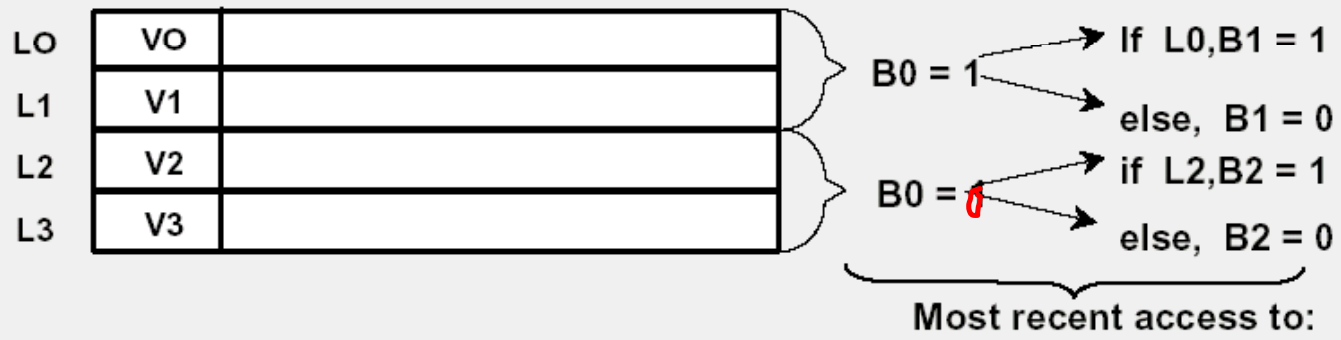
Cache: M-way Set Associative

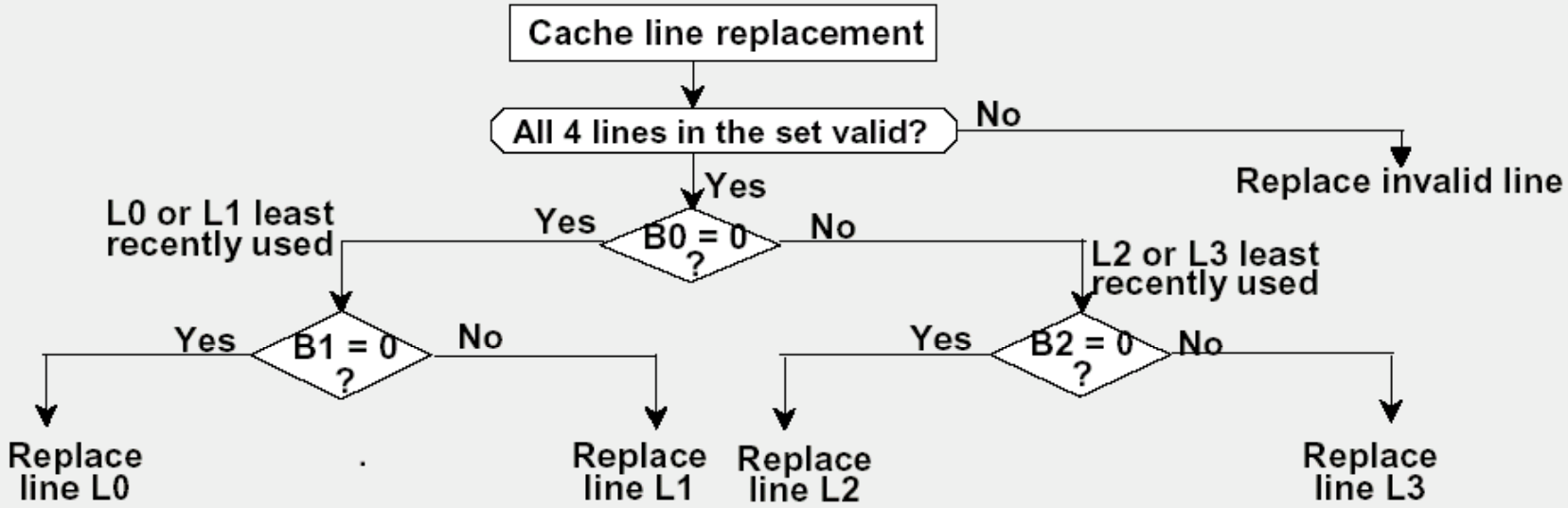
- Set associative: memory block to a set of M cache lines (A compromise between fully associative and direct mapped organizations) (指定M車位:停找均易)
  - #sets = #cache_lines/M (-way)
    - e.g., 2-way into 8 lines => 4 sets (addressed by 2-bit set index)
  - #tag_bits: medium (X – set index bits)
    - Set index need not be saved as tags
      - A set index => M Cache indexes (e.g., M= lines_per_set = 2)
  - #comparators = M
    - Parallel comparison with set-index-selected M lines
  - Trashing_rate: medium

486 Processor Model: 4-way Set Associative Cache

- Replacement policy (LRU)
  - 4 valid bits: all 4 lines in use ?
    - NO => use any unused line (& save tag bits)
    - YES => find one to replace
  - LRU bits: which is least recently used
    - B0, B1, B2 => L0 ~ L3
    - B0=0 => L0/L1, if B1=0 => L0, else L1
    - B0=1 => L2/L3, if B2=0 => L2, else L3
  - Burst mode to fill cache line: 4 bytes per cycle (Chapter 7)

Pseudo LRU (least recently used) algorithm





Pentium Review …

- Block diagram: BIU, CPU, FPU, MMU, cache
- U, v pipelines

Pentium: Superscaler Processor

- available in 1992
- 32-bit architecture
- Superscaler architecture
  - Scaling: scaling down etchable feature size to increase complexity of IC (e.g., DRAM)
    - 10 microns/4004 to 0.13 microns (2001)
  - Superscaler: go beyond simply scaling down
  - Two instruction pipelines: each with own ALU, address generation circuitry, data cache interface
  - Execute two different instructions simultaneously

Pentium: Superscaler Processor

- Onboard cache
  - Separate 8K data and code caches to avoid access conflicts
- FPP
- Instruction pipeline: 8 stage
- Optimized floating point functions
  - 5x-10x FLOP's of 486
  - 2x performance of 486 at any clock rate

Pentium: Superscaler Processor

- Compatibility with 386/486:
  - Internal 32-bit registers and address bus
  - Data bus expanded to *64-bits* for higher data transfer rate
    - Compare 8088 to 386sx transition

Pentium: Superscaler Processor

- non-clone competition from AMD, Cyrix
- development of brand identity by Intel

Pentium Model …

- Block diagram: (3.23)
  - Bus interface: 64-bit data bus (was 32)
    - Burst mode of transfer for fast cache fill
    - **BTB**: pre-fetch for jumped instructions
  - CPU, MMU
  - FPU: 8-stage pipeline
  - Cache units:
    - 32 bytes (was 16)
    - 8K data + 8K code
    - 2-way (not 4-way) set associative
    - Higher hit rate (32-byte lines)
    - Higher trashing (2-way, not 4-way)

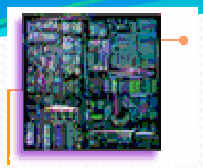
Pentium Model …

- Operations of *u* & *v* pipelines
  - *u*: all instructions
  - *v*: simple integer instructions
  - Pre-fetcher sort the incoming instructions
  - 2 simple instructions per clock cycle (in parallel)
- Versions of processors

Pentium Pro Model …

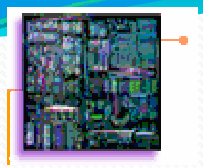
- Processing cycles against sequential fetch+execution processors
- Block diagram:
  - BIU, CPU, MMU, FPU, cache, APIC
- Comparison in performance

Pentium Pro: Two Chips in One



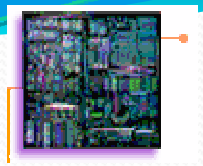
- Became available in 1995
- Superscaler of degree 3
  - Can execute 3 instructions simultaneously
- Optimized for 32-bit operating systems (e.g., Windows NT, OS2/Warp)
- Two separate silicon die on the same package
  - Processor: 0.35 u, 5.5 million transistors
  - 256KB(/512K) Level 2 cache included on chip, 15.5 million transistors in smaller area

Pentium Pro: Two Chips in One



- On Board Level 2 cache
  - Simplifies system board design
  - Requires less space
  - Gains faster communication with processor
- Internal (level 1) cache: 8K
- Pentium Pro 133 ~= 2x Pentium 66 ~= 4x 486DX2 66

Pentium Pro:Dynamic Execution



- Dynamic execution: reduce idle processor time by predicting instruction behaviors
  - Multiple Branch Prediction: look as far as 30 instructions ahead to anticipate program branches
  - Data Flow Analysis: looks at upcoming instructions and determine if they are available for processing, depending on other instructions. Determine optimal execution sequences.
  - Speculative Execution: execute instructions in different order as entered. Speculative results are stored until final states can be determined.

68000 MICROPROCESSOR

Overview of The 68000

- The 68000 has 17 general-purpose register, each 32 bits long, plus a 32-bit program counter and a 16-bit status register. Eight of the general-purpose registers are used as data registers for byte, word, and long-word operations. The other nine general-purpose registers are address registers, which can function as stack pointers and base address registers. All 17 general-purpose registers can serve as index registers.
- The software capabilities of the 68000 are impressive by any standard, and reflect the fact that this microprocessor has been designed by programmers, for programmers.
- The 68000 can operate on five different types of data - bits,4-bit binary-coded-decimal (BCD) digits, 8-bit bytes, 16 bit words, and 32-bit long words. Byte data may be addressed on even- or odd- address boundaries, whereas word and long-word data must only be addressed on even-address boundaries.

Instruction set

- The 68000 is actually a 32-bit architecture internally, but 16-bit externally. It has 24-bit addressing and a linear address space, with none of the evil segment registers of Intel's contemporary processors that make programming them unpleasant. That means that a single directly accessed array or structure can be larger than 64KB in size. Addresses are computed as 32 bit, but the top 8 bits are cut to fit the address bus into a 64-pin package (address and data share a bus in the 40 pin packages of the and Zilog Z800).

- The 68000 has an orthogonal instruction set and sixteen registers, split into data and address registers. One address register is reserved for the Stack Pointer. Both types of registers can be used for any function except direct addressing. Only address registers can be used as the source of an address, but data registers can provide the offset from an address.

More Overview

- There are two operating modes in the 68000: User and Supervisor. Certain instructions in supervisor mode are not available in user mode. The supervisor mode is a protection against operator misuse, in sophisticated, multitasking systems. It should be interesting to see what Atari does with the supervisor mode.

  Other niceties include built-in debugging aids, traps against illegal addressing and illegal instructions, a one-step trace mode, and seven levels of vectored interrupts. Most of these are only available from the supervisory mode.
- Although the 68000 has a 16-bit data bus, meaning that 2 bytes of information can be accessed in one machine cycle, internally it can operate on five different types of data: bits, 4-bit binary coded decimal (BCD), 8-bit bytes (B), 16-bit words (W), and 32-bit long words (L). Because of this, byte data may be addressed at even or odd addresses, but words and long words must be addressed at even addresses. For example, three bytes in a row could fall at addresses $0004, $0005 and $0006, three words at $0004, $0006, $0008, and three long words at $0004, $0008, $000C.

  ((The 68000 has 56 instructions and 14 addressing modes. This is very similar to the 6502. But there are 17 general-purpose 32-bit registers. Eight are considered data registers, seven are address registers, one is the stack pointer and the last is the program counter.

Registers and Status Registers

All of the data registers are general purpose and can be used as index registers or counters. They can handle bytes, words, and long words. The address registers are primarily designed to hold addresses, but can be used as index registers. Unlike the data registers, they cannot handle 8-bit bytes.

The stack pointer can also be used as a general purpose address register. It is actually two registers and will contain different data depending upon whether you are in supervisor or user operating mode.

The last 32-bit register is the program counter and, although it is a 32-bit register, only 23 of the bits are used. Since instructions consist of words instead of bytes, the counter can access a range of 8M words, or 16,777,216 bytes.

The last register in the 68000 is the 16-bit status register, which is divided into two 8-bit bytes. The lower 8 bits are for the user mode and the upper 8 for the supervisor. Not all available bits are used. The user flag bits are:

More Registers

- The last register in the 68000 is the 16-bit status register, which is divided into two 8-bit bytes. The lower 8 bits are for the user mode and the upper 8 for the supervisor. Not all available bits are used. The user flag bits are:

BIT	SYMBOL	CONDITION
0	C	Carry
1	V	Overflow
2	Z	Zero
3	N	Negative
4	X	Extend
5-7	(Unused)	

- Supervisor status flag bits 8 through 9 are used in various combinations to signal interrupt priority for the seven levels of interrupt. The 13th bit switches the modes between supervisor and user, and the 15th bit places the 68000 in trace mode. Bits 11,12 and 14 are unused.

- 68000 Microprocessor chip hardware

The 68000 microprocessor is housed in a 64-pin dual in-line package (DIP). This convention is intended to distinguish between signals that are active in low or logic-0 state and signals that are active in the high or logic-1 state.

The 68000 microprocessor operates from +5 volts, connected to two pins labeled Vcc, and using two ground pins labeled GND. The clock inputs is a TTL-level signal that can have a frequency of up to 10 MHz.

The 68000 is called a 16-bit microprocessor because its basic unit of information, the word, is 16 bits wide. The 68000 identifies an external device by transmitting its unique address throughout the system over 23 address bus lines. The 68000 notifies all system devices that a valid address is on the address bus by asserting the address strobe signal.

Internal Architecture of the 68xxx

- They employ a high-performance, pipelined, internal architecture. That is, multiple processing units are used to implement dedicated operations concurrently. This parallel processing leads to the high performance. The microprocessor's internal architecture has bus controller and instruction cache unit, instruction prefetch and decode unit, sequencer and control unit, and execution unit.
- The bus controller and instruction cache are at the 68xxxx's interface to the outside world. Meaning, this section provides the 32-bit data over which data are transferred during read, write, and interrupt acknowledge bus cycles. The bus controller produces the control signals that are required to coordinate data transfers over the bus.
- The on-chip memory provided in the instruction cache is used to store in instructions that were most recently fetched from the main memory. That is, a small segment of code is to be reexecuted while it is still in the cache, the instructions held in the cache will be used instead of requiring the instructions to be refetched from the cache.

Internal Architecture

- Instruction prefetch and decode unit provides the mechanism for simultaneous fetch, decode, and execution of instructions. At a given time, the processor may be working on three words of an instruction or three separate sequential word-sized instructions.
- The objective of this part of the pipelined architecture is to eliminate the amount of time in an instruction's execution that represented the fetch and decode operations. This can be achieved because the operations are done while the previous instructions is still being executed. This makes it fast.
- Execution unit is to read decoded instructions from the output of the decode unit and perform the operations defined by the instructions. It contains the ALU.
- Sequencer and control units coordinate the overall operations of all the processing units.