IA-32 Architecture

Computer Organization and Assembly Languages

Yung-Yu Chuang

2006/10/30

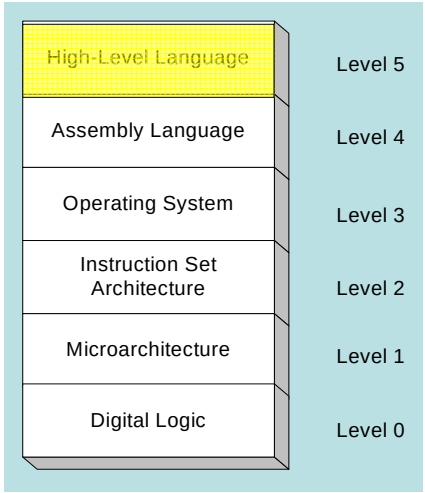
with slides by Kip Irvine, Robert Sedgwick and Kevin Wayne

Announcements

- Midterm exam date. 11/13 (specified by school) or 11/20?
- Open-book

Virtual machines

Abstractions for computers

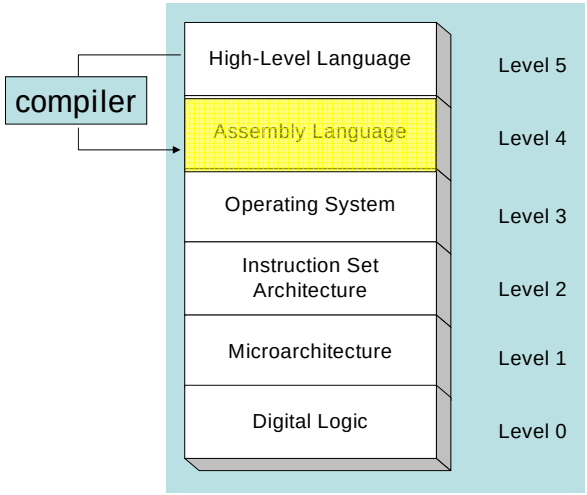


High-level language

```
int A[32];



i=0;
Do {
  r<stdin;
  if (r==0)
  break;

  A[i]=r;
  i=i+1;
} while (1);

printr();
```

Virtual machines

Abstractions for computers

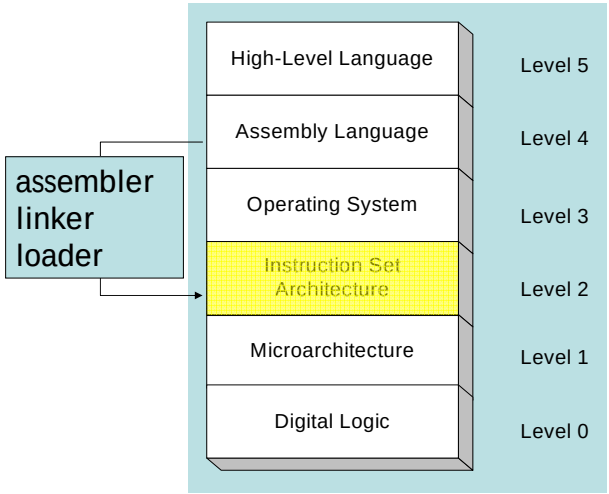


Assembly language

```
int A[32];          A     DUP   32

                          lda   R1, 1
                          lda   RA, A
i=0;                      lda   RC, 0
Do {
  r<stdin;          read  ld    RD, 0xFF
  if (r==0)               bz    RD, exit
  break;                  add   R2, RA, RC
                          sti   RD, R2
  A[i]=r;                 add   RC, RC, R1
  i=i+1;                  bz    R0, read
} while (1);
                    exit  jl    RF, printr
printr();                 hlt
```

Virtual machines

Abstractions for computers

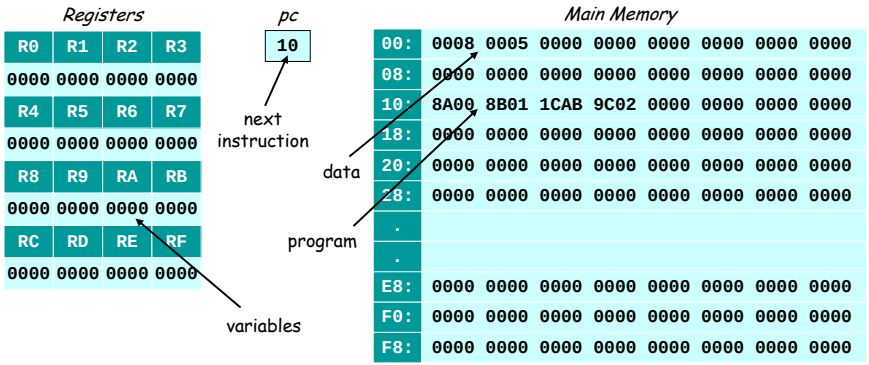


Instruction set architecture

```
int A[32];          A     DUP   32              10: C020

                          lda   R1, 1           20: 7101
                          lda   RA, A           21: 7A00
i=0;                      lda   RC, 0           22: 7C00
Do {
  r<stdin;          read  ld    RD, 0xFF        23: 8DFF
  if (r==0)               bz    RD, exit        24: CD29
  break;                  add   R2, RA, RC      25: 12AC
                          sti   RD, R2          26: BD02
  A[i]=r;                 add   RC, RC, R1      27: 1CC1
  i=i+1;                  bz    R0, read        28: C023
} while (1);
                    exit  jl    RF, printr      29: FF2B
printr();                 hlt                   2A: 0000
```

Instruction set architecture

- Machine contents at a particular place and time.
  - Record of what program has done.
  - Completely determines what machine will do.



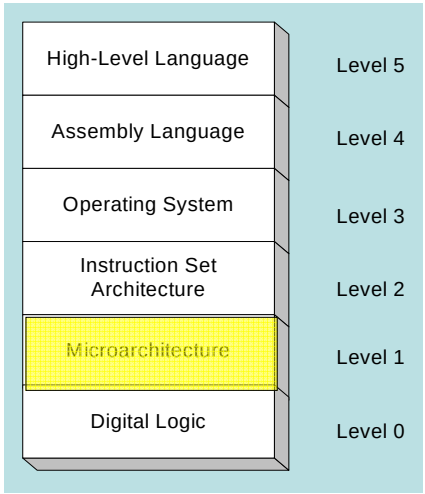
Instruction set architecture

#	Operation	Fmt	Pseudocode
0:	halt	1	exit(0)
1:	add	1	R[d] ← R[s] + R[t]
2:	subtract	1	R[d] ← R[s] - R[t]
3:	and	1	R[d] ← R[s] & R[t]
4:	xor	1	R[d] ← R[s] ^ R[t]
5:	shift left	1	R[d] ← R[s] << R[t]
6:	shift right	1	R[d] ← R[s] >> R[t]
7:	load addr	2	R[d] ← addr
8:	load	2	R[d] ← mem[addr]
9:	store	2	mem[addr] ← R[d]
A:	load indirect	1	R[d] ← mem[R[t]]
B:	store indirect	1	mem[R[t]] ← R[d]
C:	branch zero	2	if (R[d] == 0) pc ← addr
D:	branch positive	2	if (R[d] > 0) pc ← addr
E:	jump register	2	pc ← R[d]
F:	jump and link	2	R[d] ← pc; pc ← addr

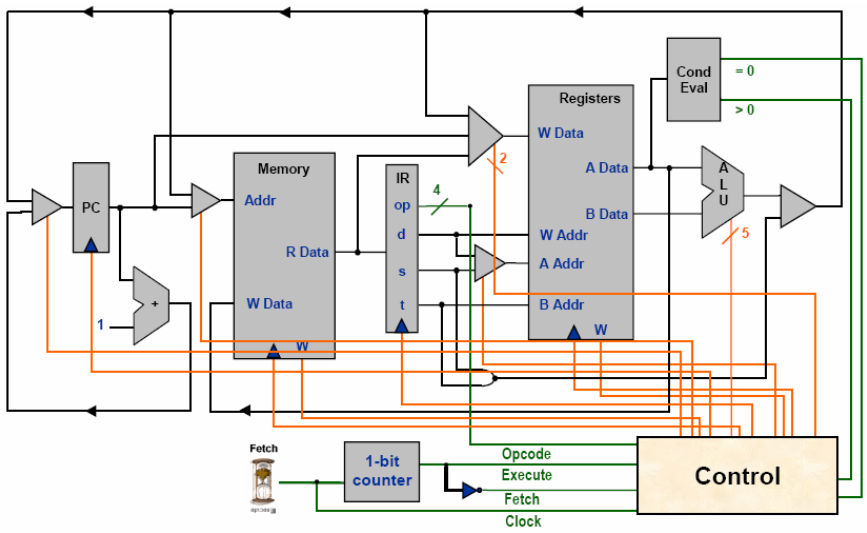
Register 0 always 0.
Loads from mem[FF] from stdin.
Stores to mem[FF] to stdout.

Virtual machines

Abstractions for computers

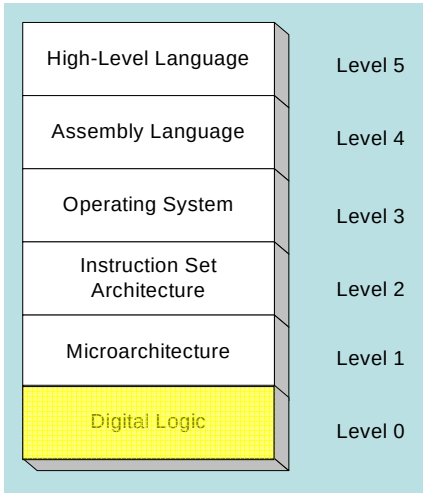


Architecture

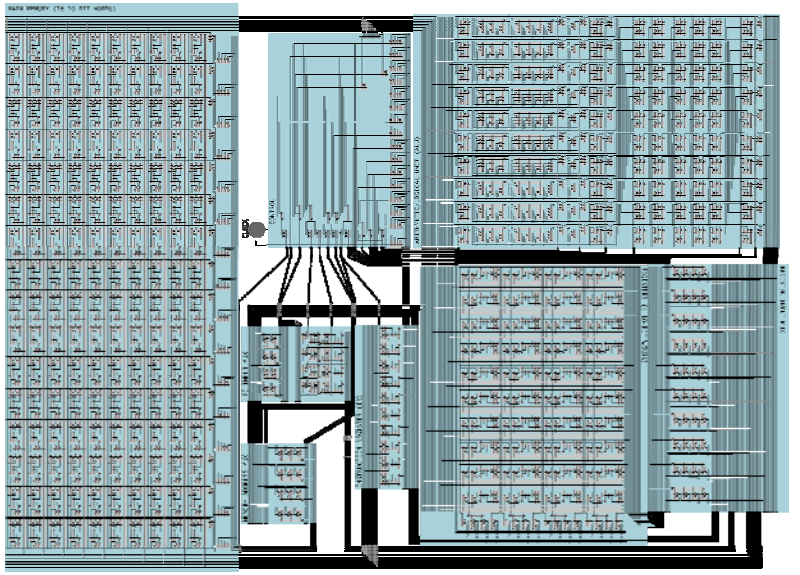


Virtual machines

Abstractions for computers



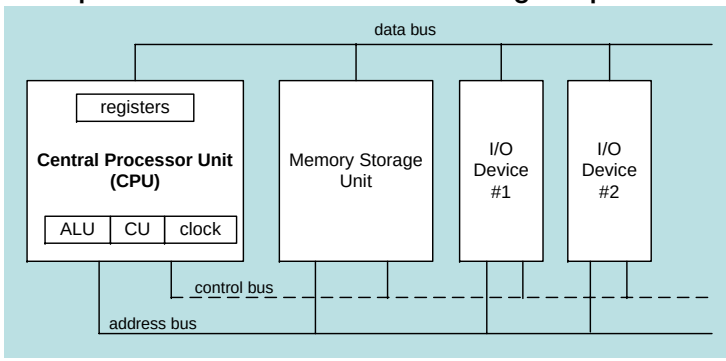
Gate level



Basic architecture

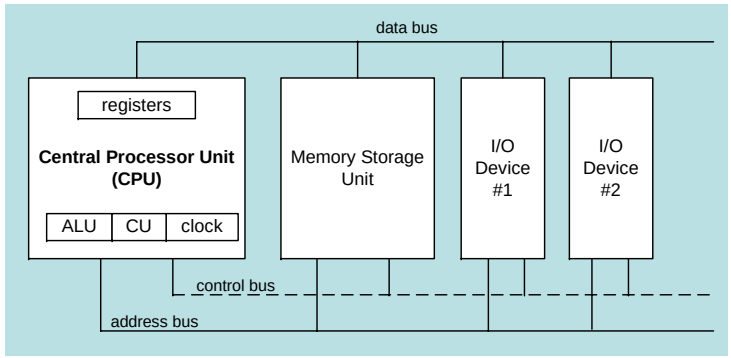
Basic microcomputer design

- clock synchronizes CPU operations
- control unit (CU) coordinates sequence of execution steps
- ALU performs arithmetic and logic operations



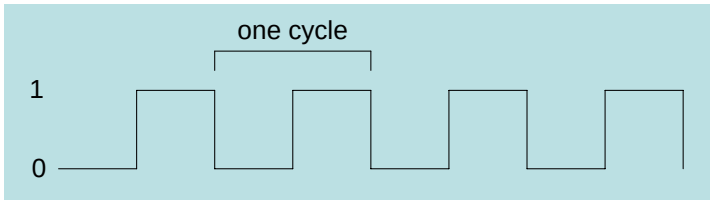
Basic microcomputer design

- The memory storage unit holds instructions and data for a running program
- A bus is a group of wires that transfer data from one part to another (data, address, control)



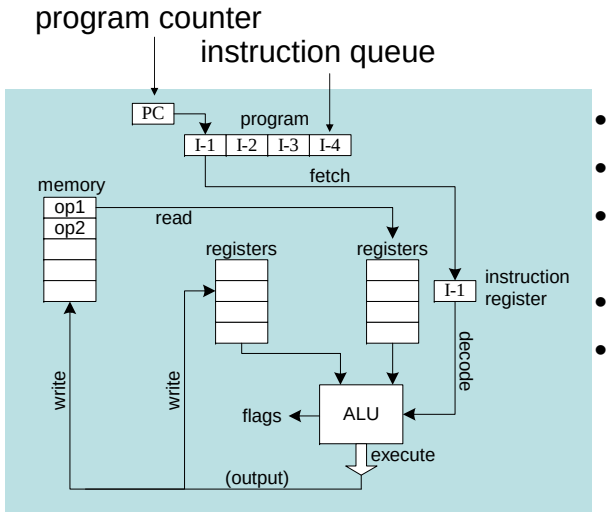
Clock

- synchronizes all CPU and BUS operations
- machine (clock) cycle measures time of a single operation
- clock is used to trigger events



- Basic unit of time, 1GHz→clock cycle=1ns
- A instruction could take multiple cycles to complete, e.g. multiply in 8088 takes 50 cycles

Instruction execution cycle



- **Fetch**
- Decode
- Fetch operands
- **Execute**
- Store output

Advanced architecture

Multi-stage pipeline

- Pipelining makes it possible for processor to execute instructions in parallel
- Instruction execution divided into discrete stages

Example of a non-pipelined processor. For example, 80386. Many wasted cycles.

Stages

Cycles	S1	S2	S3	S4	S5	S6
1	I-1					
2		I-1				
3			I-1			
4				I-1		
5					I-1	
6						I-1
7	I-2					
8		I-2				
9			I-2			
10				I-2		
11					I-2	
12						I-2

Pipelined execution

- More efficient use of cycles, greater throughput of instructions: (80486 started to use pipelining)

Stages

Cycles	S1	S2	S3	S4	S5	S6
1	I-1					
2	I-2	I-1				
3		I-2	I-1			
4			I-2	I-1		
5				I-2	I-1	
6					I-2	I-1
7						I-2

For *k* stages and *n* instructions, the number of required cycles is:

$k + (n - 1)$

compared to k*n

Wasted cycles (pipelined)

- When one of the stages requires two or more clock cycles, clock cycles are again wasted.

Stages

Cycles	S1	S2	S3	exe S4	S5	S6
1	I-1					
2	I-2	I-1				
3	I-3	I-2	I-1			
4		I-3	I-2	I-1		
5			I-3	I-1		
6				I-2	I-1	
7				I-2		I-1
8				I-3	I-2	
9				I-3		I-2
10					I-3	
11						I-3

For *k* stages and *n* instructions, the number of required cycles is:

$k + (2n - 1)$

Superscalar

A superscalar processor has multiple execution pipelines. In the following, note that Stage S4 has left and right pipelines (u and v).

Stages

Cycles	S1	S2	S3	S4 u	S4 v	S5	S6
1	I-1						
2	I-2	I-1					
3	I-3	I-2	I-1				
4	I-4	I-3	I-2	I-1			
5		I-4	I-3	I-1	I-2		
6			I-4	I-3	I-2	I-1	
7				I-3	I-4	I-2	I-1
8					I-4	I-3	I-2
9						I-4	I-3
10							I-4

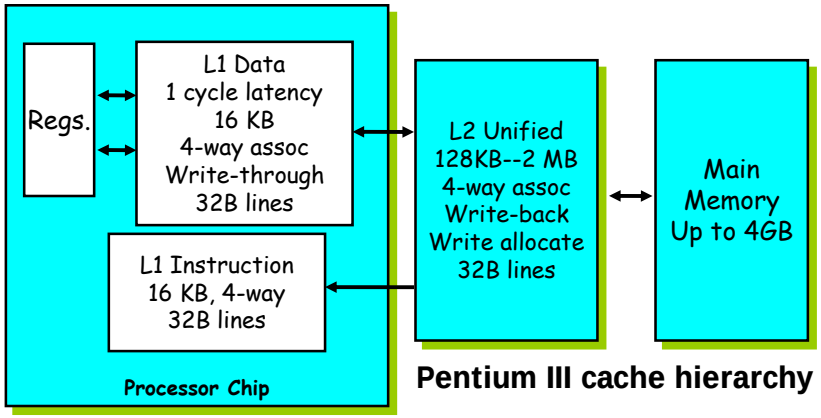
For *k* states and *n* instructions, the number of required cycles is:

$k + n$

Pentium: 2 pipelines
Pentium Pro: 3

Reading from memory

- Multiple machine cycles are required when reading from memory, because it responds much more slowly than the CPU (e.g.33 MHz). The wasted clock cycles are called wait states.

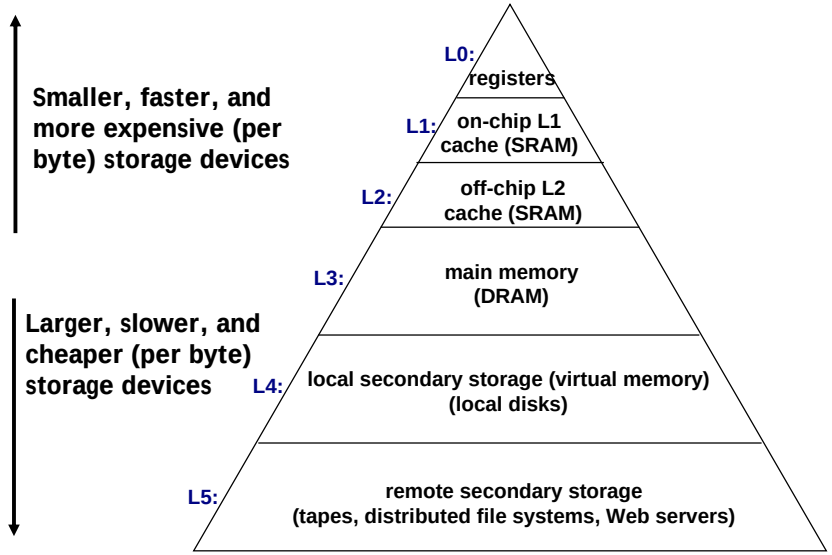


Pentium III cache hierarchy

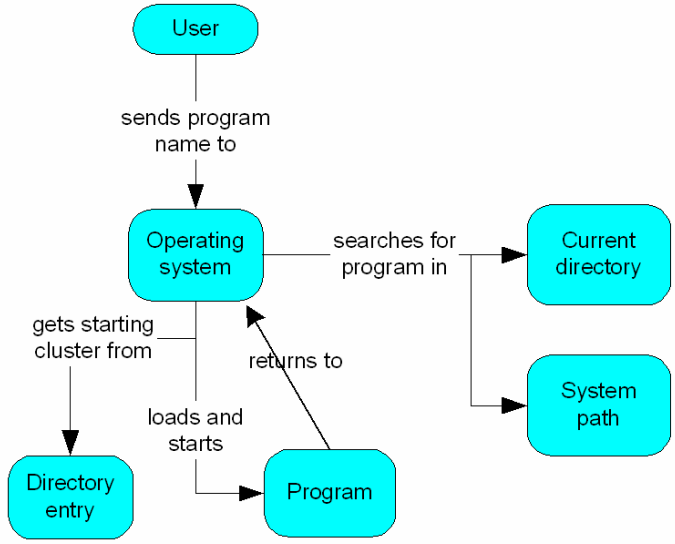
Cache memory

- High-speed expensive static RAM both inside and outside the CPU.
  - Level-1 cache: inside the CPU
  - Level-2 cache: outside the CPU
- Cache hit: when data to be read is already in cache memory
- Cache miss: when data to be read is not in cache memory. When? compulsory, capacity and conflict.
- Cache design: cache size, n-way, block size, replacement policy

Memory system in practice



How a program runs



Multitasking

- OS can run multiple programs at the same time.
- Multiple threads of execution within the same program.
- Scheduler utility assigns a given amount of CPU time to each running program.
- Rapid switching of tasks
  - gives illusion that all programs are running at once
  - the processor must support task switching
  - scheduling policy, round-robin, priority

IA-32 Architecture

IA-32 architecture

- From 386 to the latest 32-bit processor, P4
- Lots of architecture improvements, pipelining, superscalar, branch prediction and hyperthreading.
- From programmer's point of view, IA-32 has not changed substantially except the introduction of a set of high-performance instructions

Modes of operation

- Protected mode
  - native mode (Windows, Linux), full features, separate memory

  - Virtual-8086 mode
    - hybrid of Protected
    - each program has its own 8086 computer

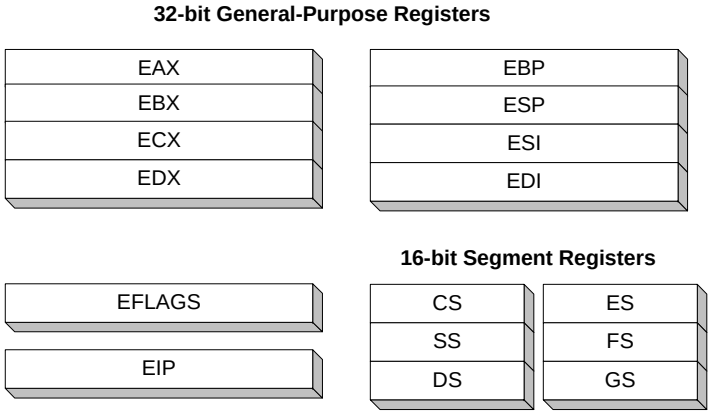
- Real-address mode
  - native MS-DOS
- System management mode
  - power management, system security, diagnostics

Addressable memory

- Protected mode
  - 4 GB
  - 32-bit address
- Real-address and Virtual-8086 modes
  - 1 MB space
  - 20-bit address

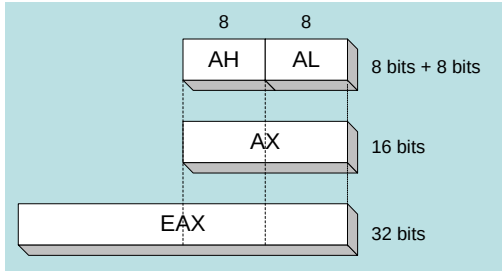
General-purpose registers

Named storage locations inside the CPU, optimized for speed.



Accessing parts of registers

- Use 8-bit name, 16-bit name, or 32-bit name
- Applies to EAX, EBX, ECX, and EDX



32-bit	16-bit	8-bit (high)	8-bit (low)
EAX	AX	AH	AL
EBX	BX	BH	BL
ECX	CX	CH	CL
EDX	DX	DH	DL

Index and base registers

- Some registers have only a 16-bit name for their lower half. The 16-bit registers are usually used only in real-address mode.

32-bit	16-bit
ESI	SI
EDI	DI
EBP	BP
ESP	SP

Some specialized register uses (1 of 2)

- General-Purpose
  - EAX – accumulator (automatically used by division and multiplication)
  - ECX – loop counter
  - ESP – stack pointer (should never be used for arithmetic or data transfer)
  - ESI, EDI – index registers (used for high-speed memory transfer instructions)
  - EBP – extended frame pointer (stack)

Some specialized register uses (2 of 2)

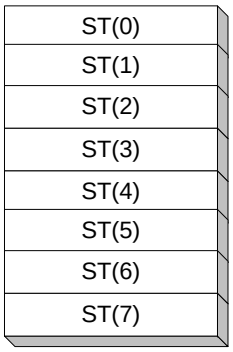
- Segment
  - CS – code segment
  - DS – data segment
  - SS – stack segment
  - ES, FS, GS - additional segments
- EIP – instruction pointer
- EFLAGS
  - status and control flags
  - each flag is a single binary bit (*set* or *clear*)

Status flags

- Carry
  - unsigned arithmetic out of range
- Overflow
  - signed arithmetic out of range
- Sign
  - result is negative
- Zero
  - result is zero
- Auxiliary Carry
  - carry from bit 3 to bit 4
- Parity
  - sum of 1 bits is an even number

Floating-point, MMX, XMM registers

- Eight 80-bit floating-point data registers
  - ST(0), ST(1), . . . , ST(7)
  - arranged in a stack
  - used for all floating-point arithmetic
- Eight 64-bit MMX registers
- Eight 128-bit XMM registers for single-instruction multiple-data (SIMD) operations



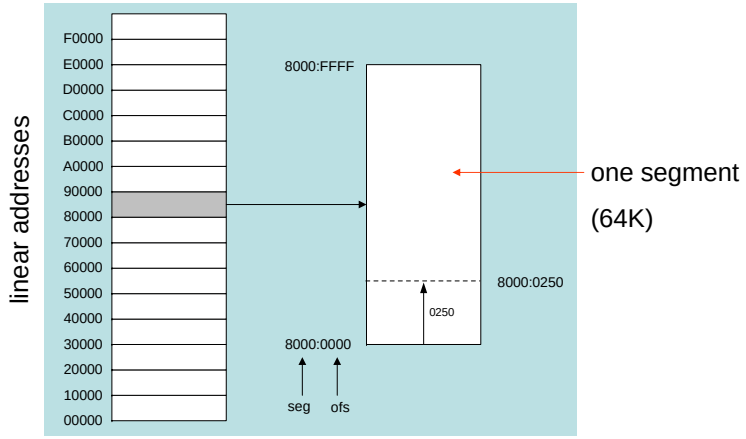
IA-32 Memory Management

Real-address mode

- 1 MB RAM maximum addressable (20-bit address)
- Application programs can access any area of memory
- Single tasking
- Supported by MS-DOS operating system

Segmented memory

Segmented memory addressing: absolute (linear) address is a combination of a 16-bit segment value added to a 16-bit offset



Calculating linear addresses

- Given a segment address, multiply it by 16 (add a hexadecimal zero), and add it to the offset
- Example: convert 08F1:0100 to a linear address

```
Adjusted Segment value:  0 8 F 1 0
Add the offset:            0 1 0 0
Linear address:          0 9 0 1 0
```

- A typical program has three segments: code, data and stack. Segment registers CS, DS and SS are used to store them separately.

Example

What linear address corresponds to the segment/offset address 028F:0030?

028F0 + 0030 = 02920

Always use hexadecimal notation for addresses.

Example

What segment addresses correspond to the linear address 28F30h?

Many different segment-offset addresses can produce the linear address 28F30h. For example:

28F0:0030, 28F3:0000, 28B0:0430, . . .

Protected mode (1 of 2)

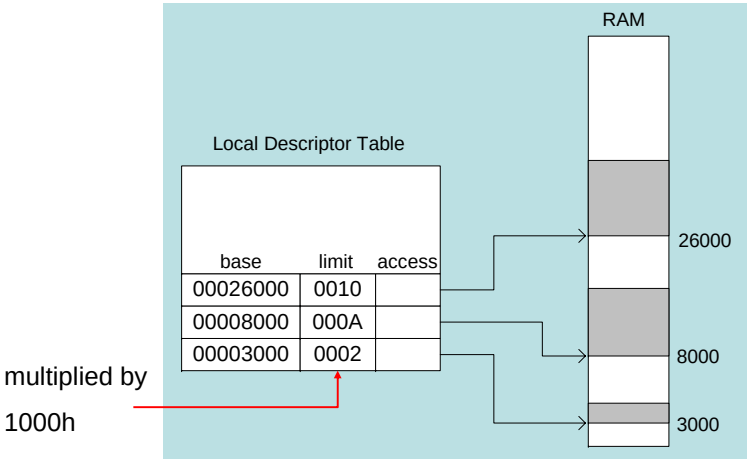
- 4 GB addressable RAM (32-bit address)
  - (00000000 to FFFFFFFFh)
- Each program assigned a memory partition which is protected from other programs
- Designed for multitasking
- Supported by Linux & MS-Windows

Protected mode (2 of 2)

- Segment descriptor tables
- Program structure
  - code, data, and stack areas
  - CS, DS, SS segment descriptors
  - global descriptor table (GDT)
- MASM Programs use the Microsoft flat memory model

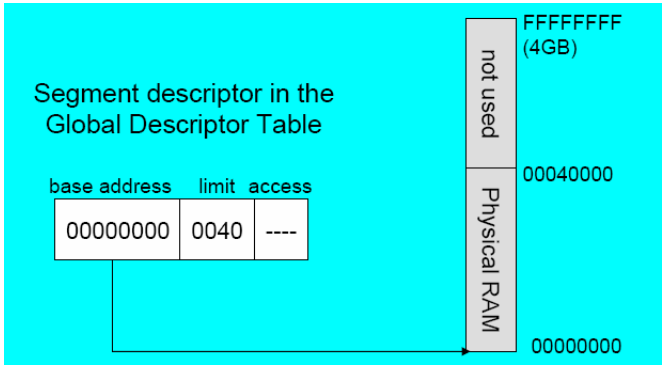
Multi-segment model

- Each program has a local descriptor table (LDT)
  - holds descriptor for each segment used by the program



Flat segmentation model

- All segments are mapped to the entire 32-bit physical address space, at least two, one for data and one for code
- global descriptor table (GDT)



Paging

- Virtual memory uses disk as part of the memory, thus allowing sum of all programs can be larger than physical memory
- Divides each segment into 4096-byte blocks called pages

- Page fault (supported directly by the CPU) – issued by CPU when a page must be loaded from disk
- Virtual memory manager (VMM) – OS utility that manages the loading and unloading of pages

Components of an IA-32 microcomputer

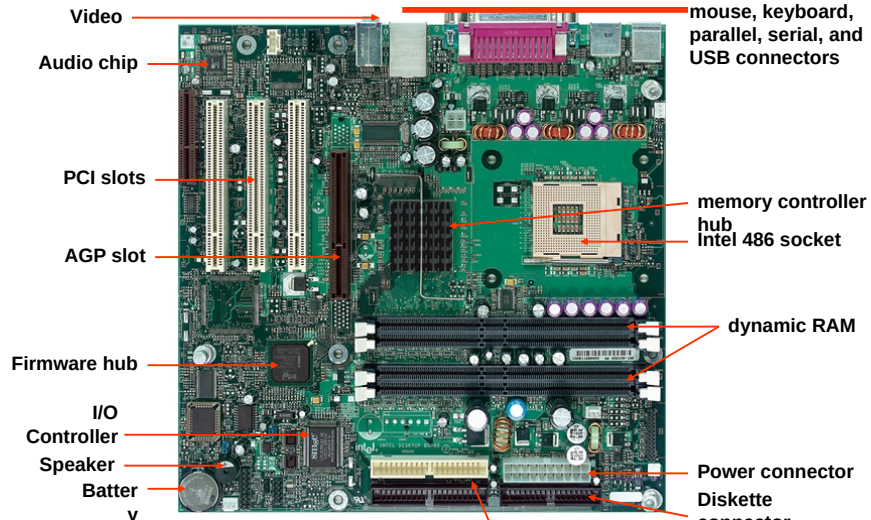
Components of an IA-32 Microcomputer

- Motherboard
- Video output
- Memory
- Input-output ports

Motherboard

- CPU socket
- External cache memory slots
- Main memory slots
- BIOS chips
- Sound synthesizer chip (optional)
- Video controller chip (optional)
- IDE, parallel, serial, USB, video, keyboard, joystick, network, and mouse connectors
- PCI bus connectors (expansion cards)

Intel D850MD motherboard



Source: Intel® Desktop Board D850MD/D850MV Technical Product Specification

Video Output

- Video controller
  - on motherboard, or on expansion card
  - AGP (accelerated graphics port)
- Video memory (VRAM)
- Video CRT Display
  - uses raster scanning
  - horizontal retrace
  - vertical retrace
- Direct digital LCD monitors
  - no raster scanning required

Memory

- ROM
  - read-only memory
- EPROM
  - erasable programmable read-only memory
- Dynamic RAM (DRAM)
  - inexpensive; must be refreshed constantly
- Static RAM (SRAM)
  - expensive; used for cache memory; no refresh required
- Video RAM (VRAM)
  - dual ported; optimized for constant video refresh
- CMOS RAM
  - refreshed by a battery
  - system setup information

Input-output ports

- USB (universal serial bus)
  - intelligent high-speed connection to devices
  - up to 12 megabits/second
  - USB hub connects multiple devices
  - *enumeration*: computer queries devices
  - supports *hot* connections
- Parallel
  - short cable, high speed
  - common for printers
  - bidirectional, parallel data transfer
  - Intel 8255 controller chip
- Serial
  - RS-232 serial port
  - one bit at a time
  - used for long cables and modems
  - 16550 UART (universal asynchronous receiver transmitter)
  - programmable in assembly language

Intel microprocessor history

Early Intel microprocessors

- Intel 8080
  - 64K addressable RAM
  - 8-bit registers
  - CP/M operating system
  - 5,6,8,10 MHz
  - 29K transistros
- Intel 8086/8088 (1978)
  - IBM-PC used 8088
  - 1 MB addressable RAM
  - 16-bit registers
  - 16-bit data bus (8-bit for 8088)
  - separate floating-point unit (8087)
  - used in low-cost microcontrollers now

The IBM-AT

- Intel 80286 (1982)
  - 16 MB addressable RAM
  - Protected memory
  - several times faster than 8086
  - introduced IDE bus architecture
  - 80287 floating point unit
  - Up to 20MHz
  - 134K transistors

Intel IA-32 Family

- Intel386 (1985)
  - 4 GB addressable RAM
  - 32-bit registers
  - paging (virtual memory)
  - Up to 33MHz
- Intel486 (1989)
  - instruction pipelining
  - Integrated FPU
  - 8K cache
- Pentium (1993)
  - Superscalar (two parallel pipelines)

Intel P6 Family

- Pentium Pro (1995)
  - advanced optimization techniques in microcode
  - More pipeline stages
  - On-board L2 cache
- Pentium II (1997)
  - MMX (multimedia) instruction set
  - Up to 450MHz
- Pentium III (1999)
  - SIMD (streaming extensions) instructions (SSE)
  - Up to 1+GHz
- Pentium 4 (2000)
  - NetBurst micro-architecture, tuned for multimedia
  - 3.8+GHz
- Pentium D (Dual core)

CISC and RISC

- CISC – complex instruction set
  - large instruction set
  - high-level operations (simpler for compiler?)
  - requires microcode interpreter (could take a long time)
  - examples: Intel 80x86 family
- RISC – reduced instruction set
  - small instruction set
  - simple, atomic instructions
  - directly executed by hardware very quickly
  - easier to incorporate advanced architecture design
  - examples:
    - ARM (Advanced RISC Machines)
    - DEC Alpha (now Compaq)