CHAPTER 2

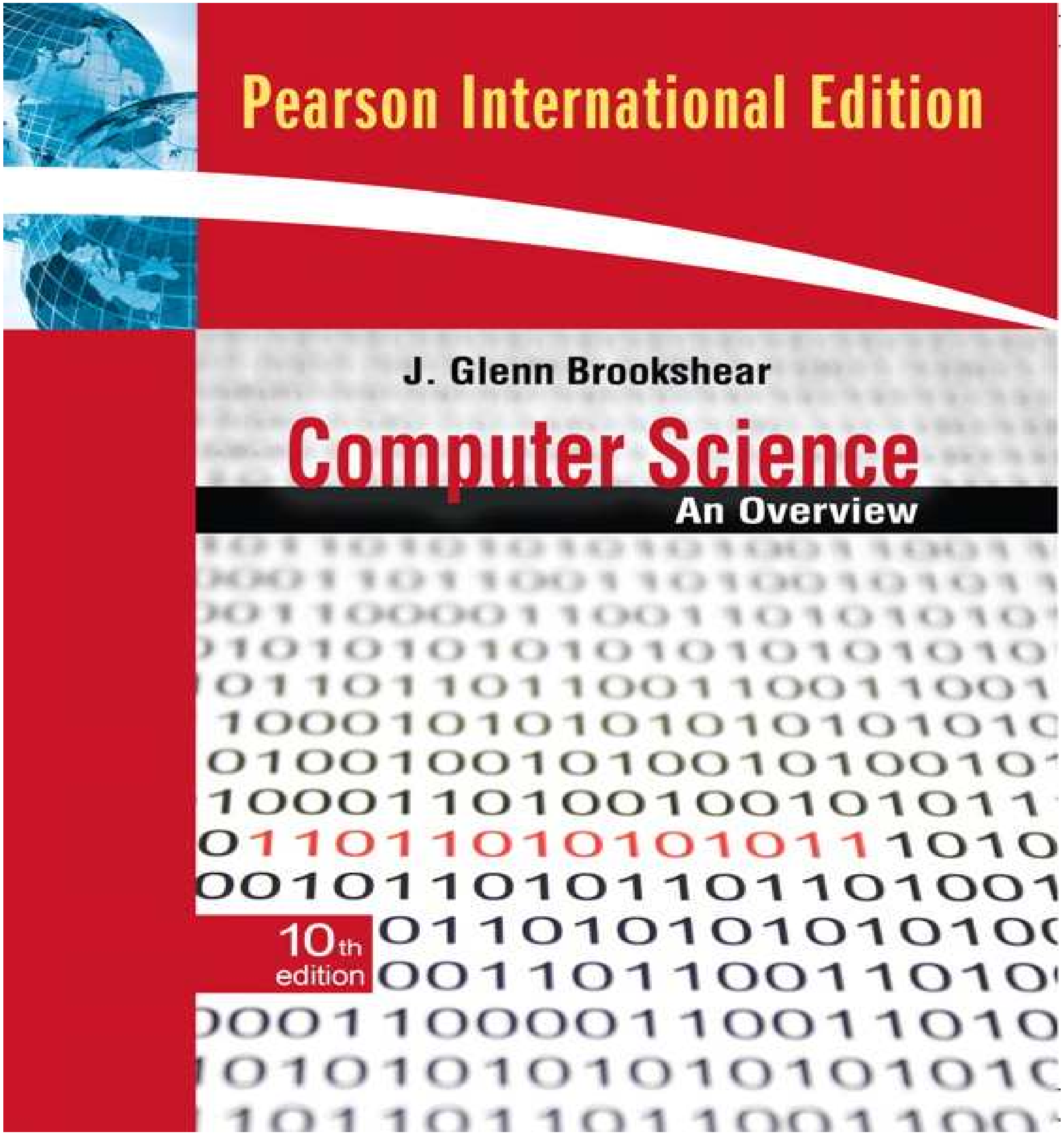
Data Manipulation

電腦如何處理資料與計算?

(請參看計概課本第二章)

蔡文能
tsaiwn@csie.nctu.edu.tw
交通大學資訊工程學系

CHAPTER 2



Pearson International Edition

J. Glenn Brookshear

Computer Science

An Overview

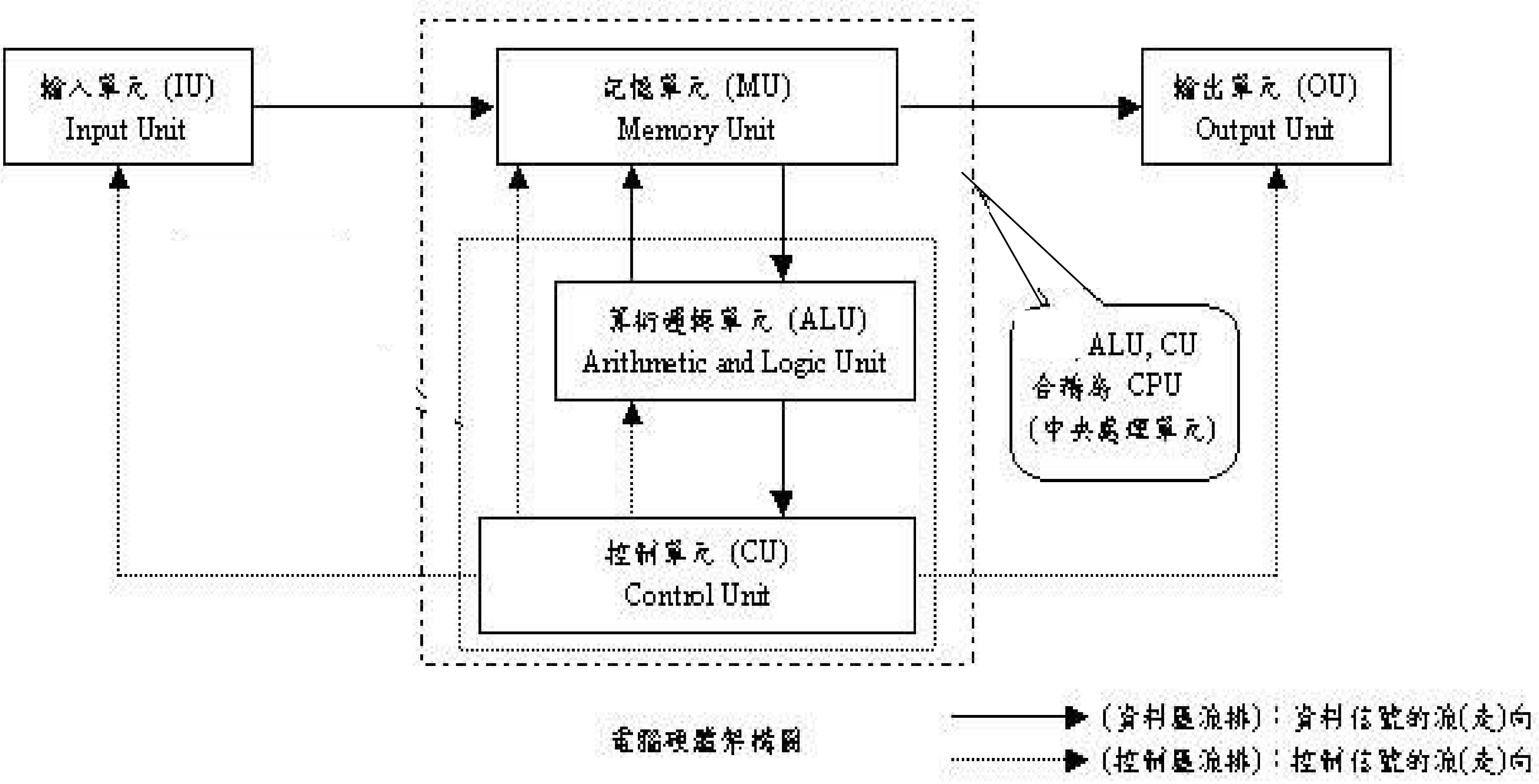
10th edition

電腦硬體五大單元　計算 2+3=?

把 2 搬到 CPU內暫存器 R5
把 3 搬到 CPU內暫存器 R6
把 R5 與 R6 加起來放到 R5

人會如何做?

電腦如何做?



電腦硬體架構圖

Figure 2.1: CPU and main memory connected via a **bus** (匯流排)

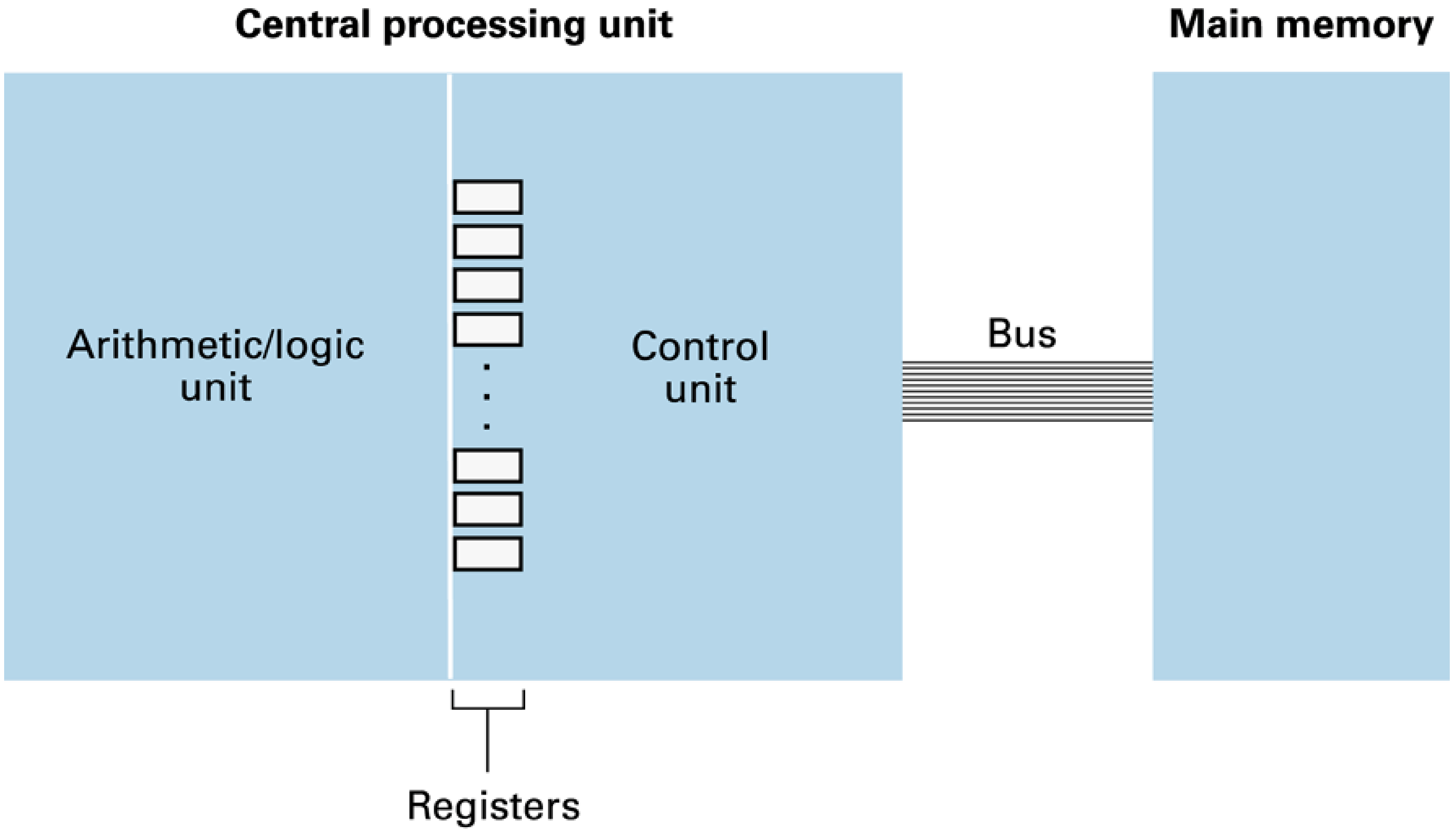
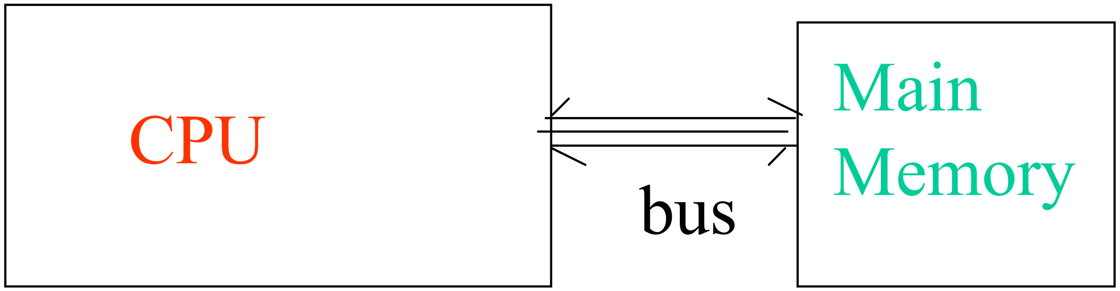


Figure 2.2: Adding values stored in memory



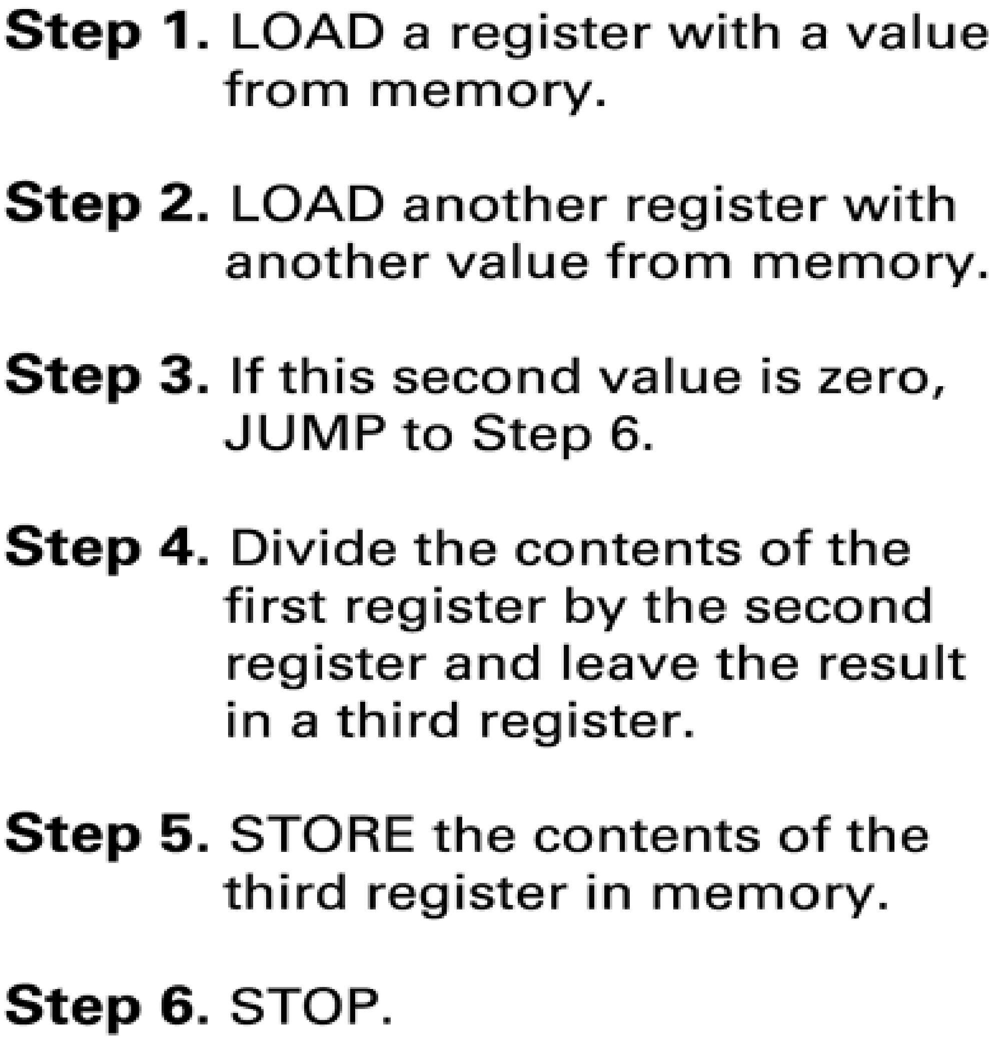
Read data by supplying memory cell address

Write data by supplying memory cell address

Example of adding 2 values

1. Get first value from memory and place in a register R1
2. Get second value from memory and place in another register R2
3. Activate addition circuitry with R1 and R2 as inputs and R3 to hold results
4. Store contents of R3 (result) in memory
5. Stop

Figure 2.3: Dividing values stored in memory



Step 1. LOAD a register with a value from memory.

Step 2. LOAD another register with another value from memory.

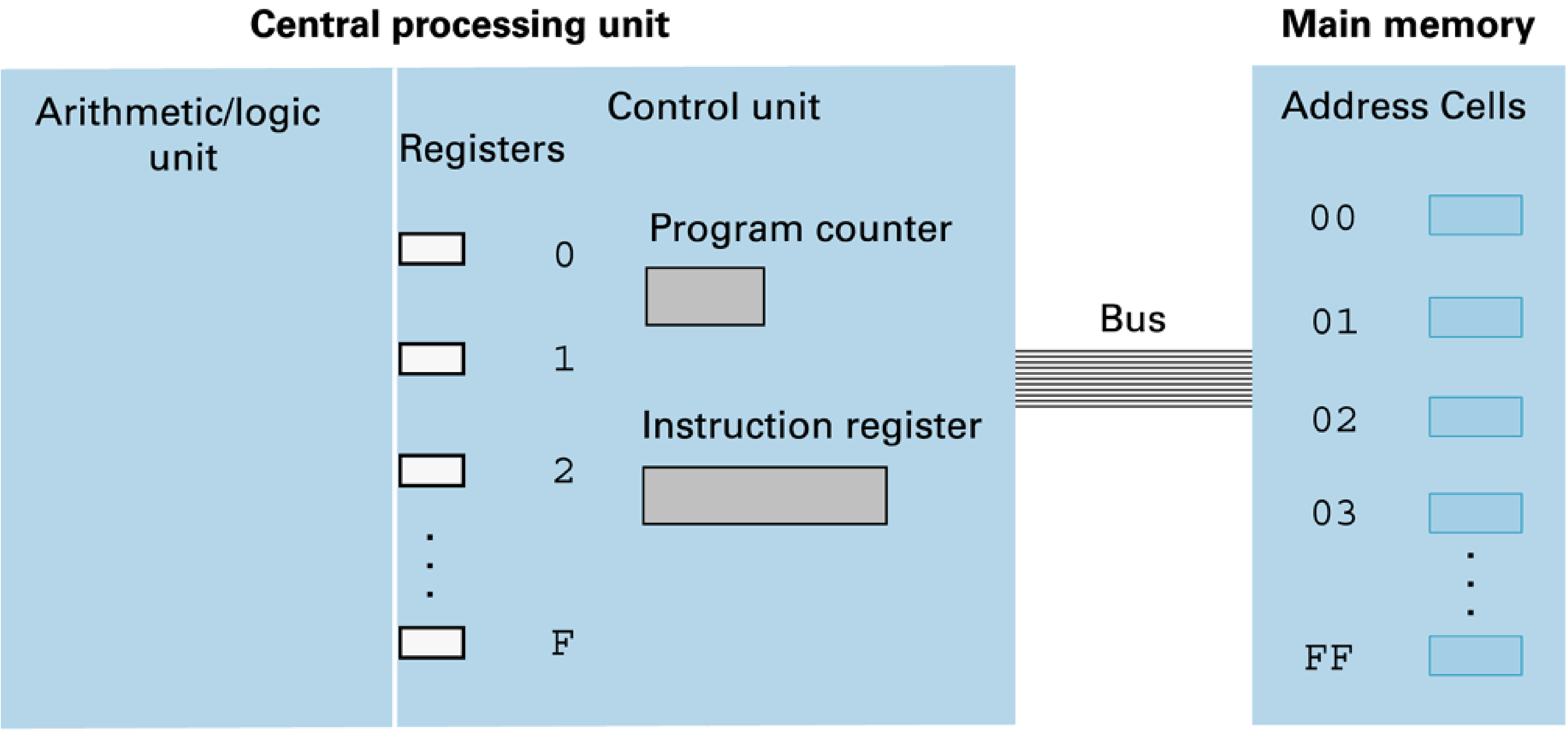
Step 3. If this second value is zero, JUMP to Step 6.

Step 4. Divide the contents of the first register by the second register and leave the result in a third register.

Step 5. STORE the contents of the third register in memory.

Step 6. STOP.

Figure 2.4: The architecture of the machine described in Appendix C



Machine Instructions1/2

Data transfer — Movement of data from one location to another

LOAD	fill a register with contents of a memory cell
STORE	transfer contents of a register to a memory cell

Arithmetic/Logic

Arithmetic operations

Logic operations AND, OR, XOR

SHIFT, ROTATE

> 其實電腦的CPU很笨
> 能做的事很有限:
> 加減, 搬來搬去, …

Control — direct execution of program

JUMP	direct control unit to execute an instruction other than the next one	
	Unconditional	Skip to step 5
	Conditional	If resulting value is 0, then skip to step 5

Machine Instructions 2/2

Example for a conditional JUMP: (因不想除以0)

1- LOAD a register R1 with a value from memory
2- LOAD register R2 with another value from memory
3- If contents of R2 is zero, JUMP to step 6

Example for a Division:

4- Divide contents of R1 by contents of R2, result stored in R3
5- STORE the content of R3 into memory
6- STOP

其實電腦的CPU很笨

能做的事很有限:加減, 搬來搬去, …

Stored-Program Concept

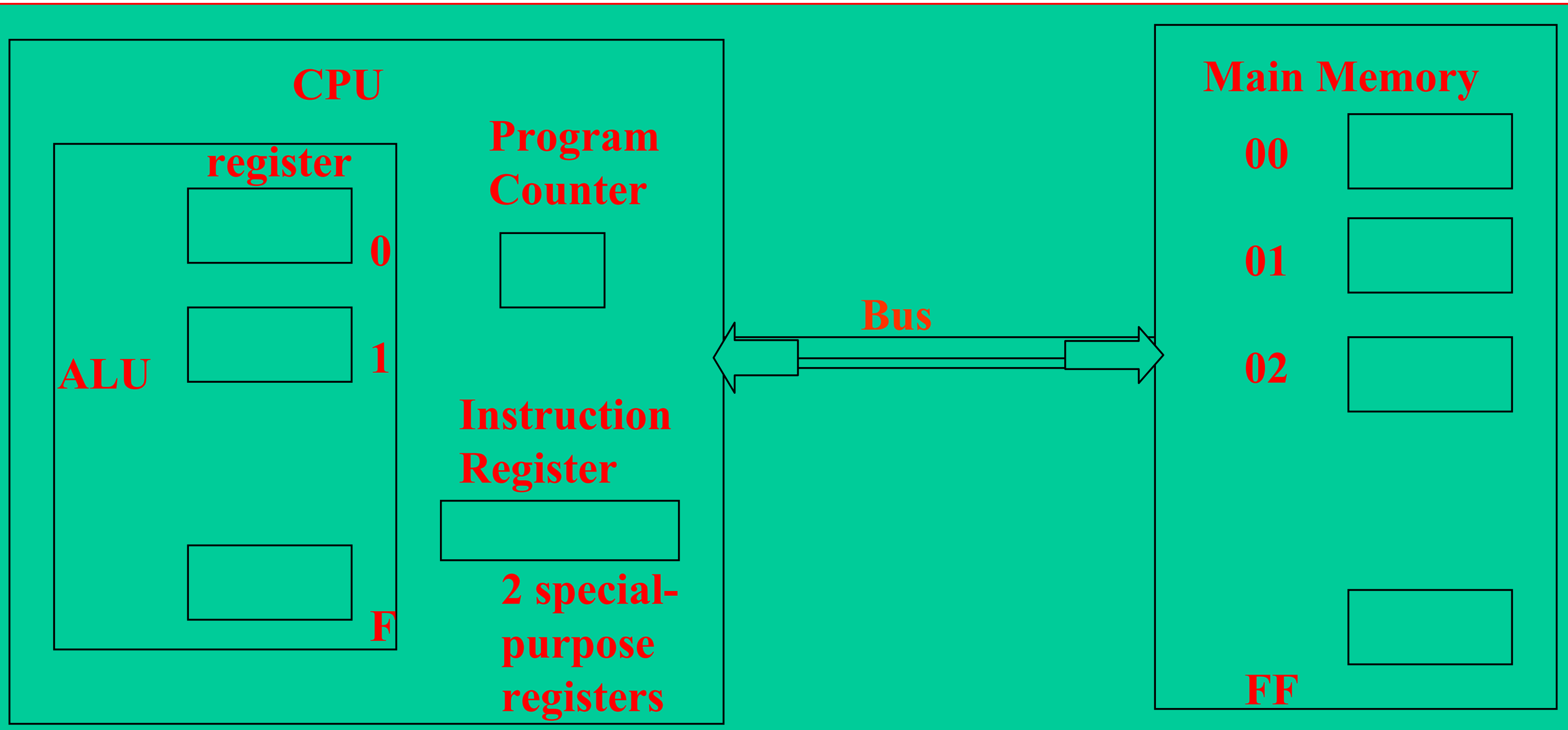
內儲程式概念：把程式碼放在 Memory讓CPU抓

- Program (instructions) stored in memory instead of being built into the control unit as part of the machine
- A machine recognizes a bit pattern as representing a specific instruction
- Instruction consists of two parts

OP-code (OPeration code)	operand field(s)

- STORE operands would be
  - ✓Register containing data to be stored
  - ✓Address of memory cell to receive data

A Sample Machine Architecture (1/2)



16 general-purpose registers

現代 CPU 還有個 SP = Stack Pointer

256 memory cells with a capacity of 8 bits each

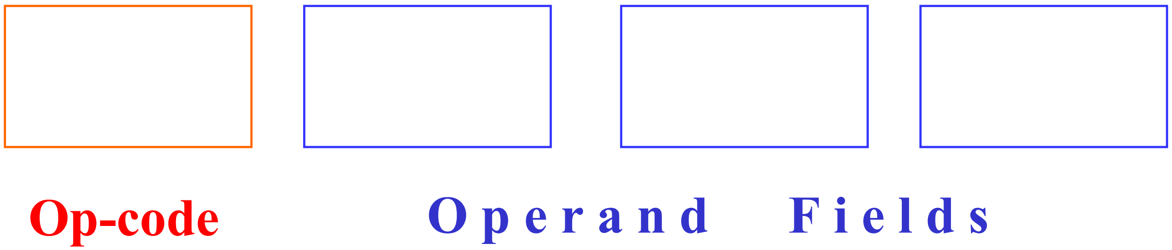
Program Counter address of next instruction to be executed

Instruction Register hold instruction being executed

A Sample Machine Architecture (2/2)

Instruction Format

Instruction consists of 4 hex digits (2 bytes)



Op-code

Operand Fields

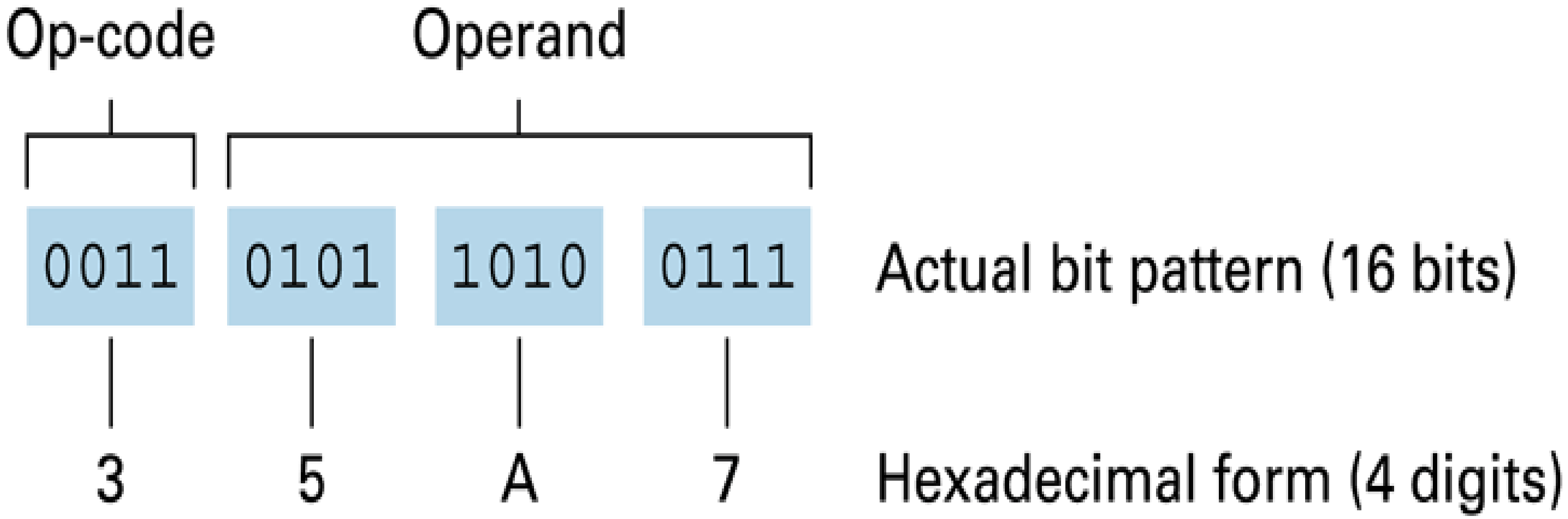
1347_{16} **LOAD register 3 with contents of the memory cell at address 47_{16}**

Textual representation might be "LOAD R3,47"

$B258_{16}$

JUMP to instruction at address 58_{16} if contents of register 2 is the same as register 0

Figure 2.5: The composition of an instruction for the machine in Appendix C



Op-code

Operand

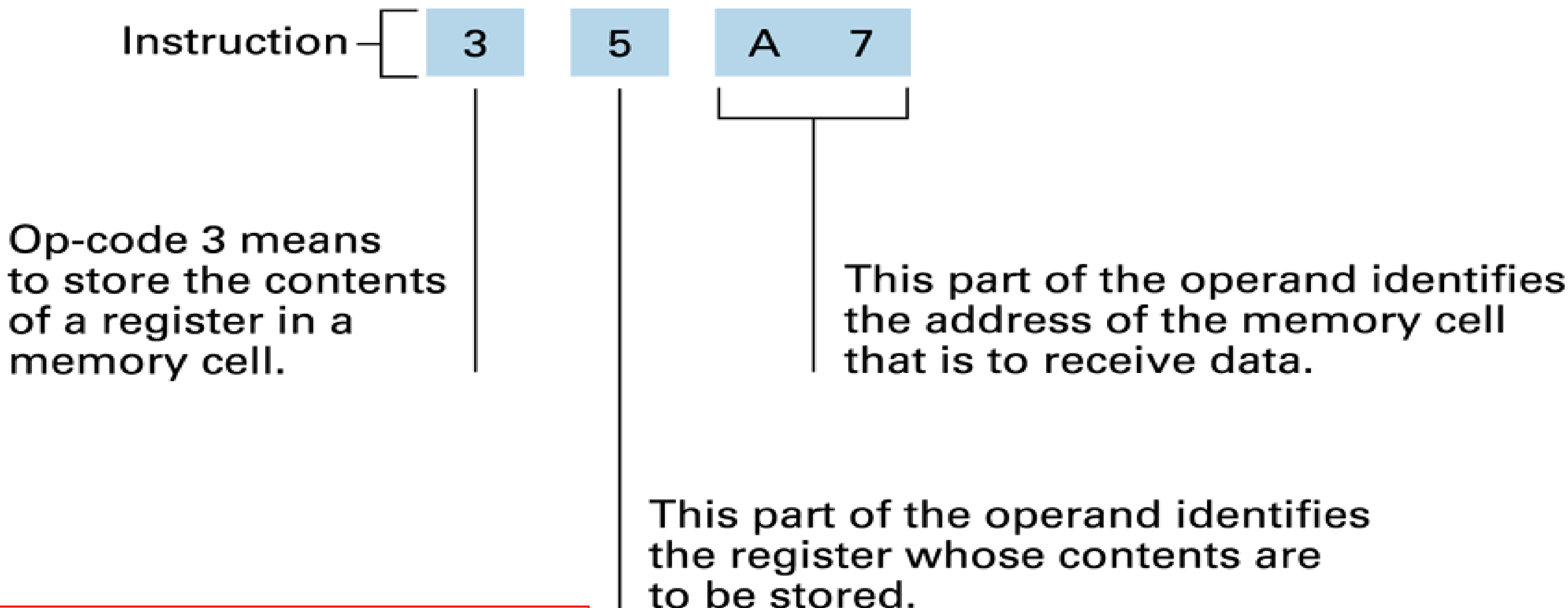
0011 0101 1010 0111 Actual bit pattern (16 bits)

3 5 A 7 Hexadecimal form (4 digits)

STORE 5, A7

See Appendix C for more instruction code

Figure 2.6: Decoding the instruction 35A7



Instruction 3 5 A 7

Op-code 3 means to store the contents of a register in a memory cell.

This part of the operand identifies the address of the memory cell that is to receive data.

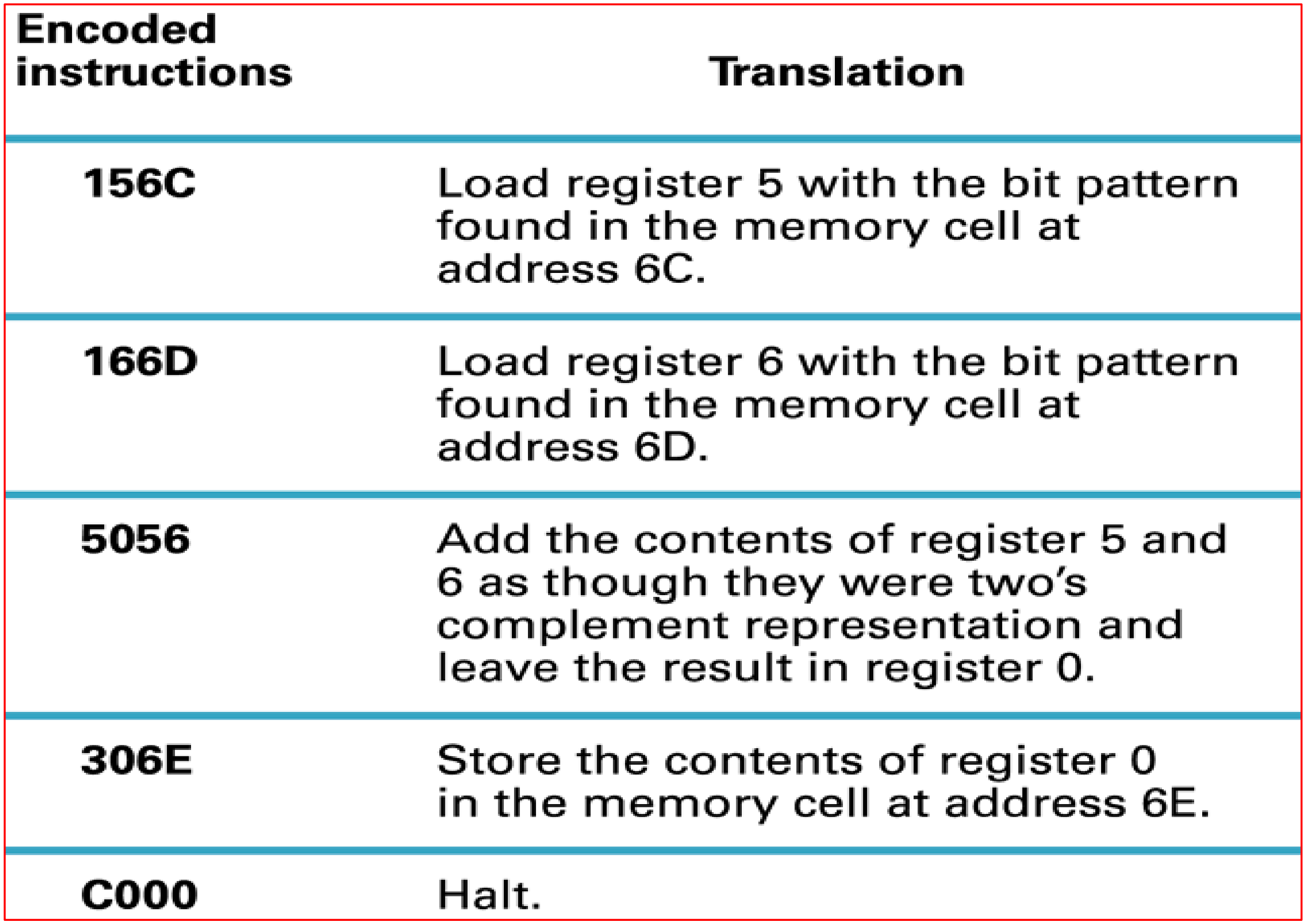
This part of the operand identifies the register whose contents are to be stored.

1 RXY = LOAD R, XY
2 RXY = LOADI R, XY
3 RXY = STORE R, XY
4 0RS = MOVE R, S
5 RST = ADD R,S,T
6 RST = ADDF R,S,T

See Appendix C

7 RST = OR R,S,T
8 RST = AND R,S,T
9 RST = XOR R,S,T
A R0X = ROR R, X
B RXY = JUMP R, XY
C 000 = HALT

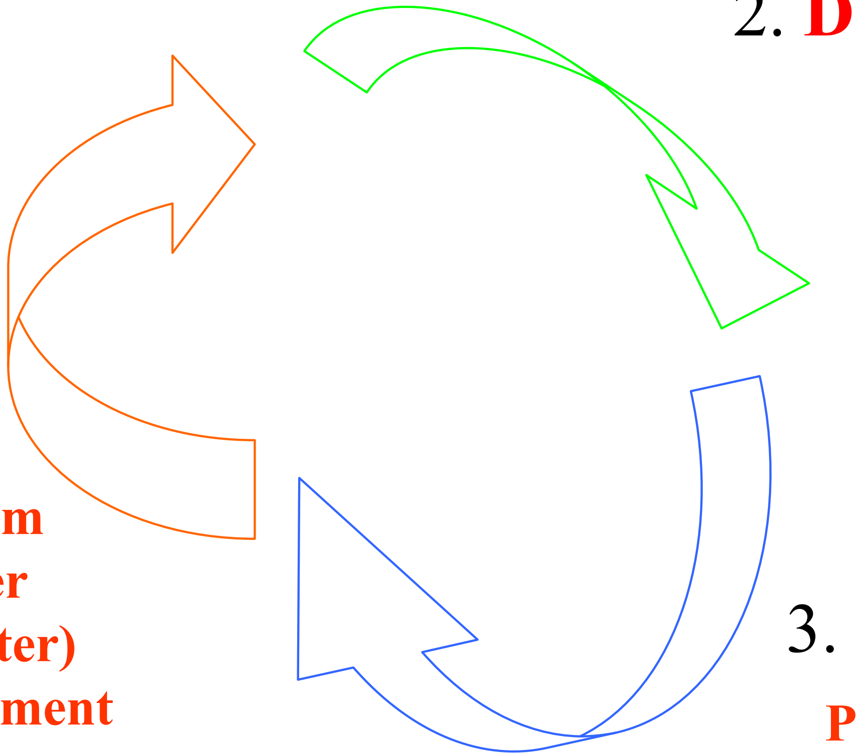
Figure 2.7: An encoded version of the Instructions in Figure 2.2



Encoded instructions	Translation
156C	Load register 5 with the bit pattern found in the memory cell at address 6C.
166D	Load register 6 with the bit pattern found in the memory cell at address 6D.
5056	Add the contents of register 5 and 6 as though they were two's complement representation and leave the result in register 0.
306E	Store the contents of register 0 in the memory cell at address 6E.
C000	Halt.

Figure 2.8: The machine cycle (Program Execution)

JUMP Instruction $B258_{16}$



1.**Fetch**

Retrieve next instruction from memory (as per program counter) and then increment program counter

2. **Decode**

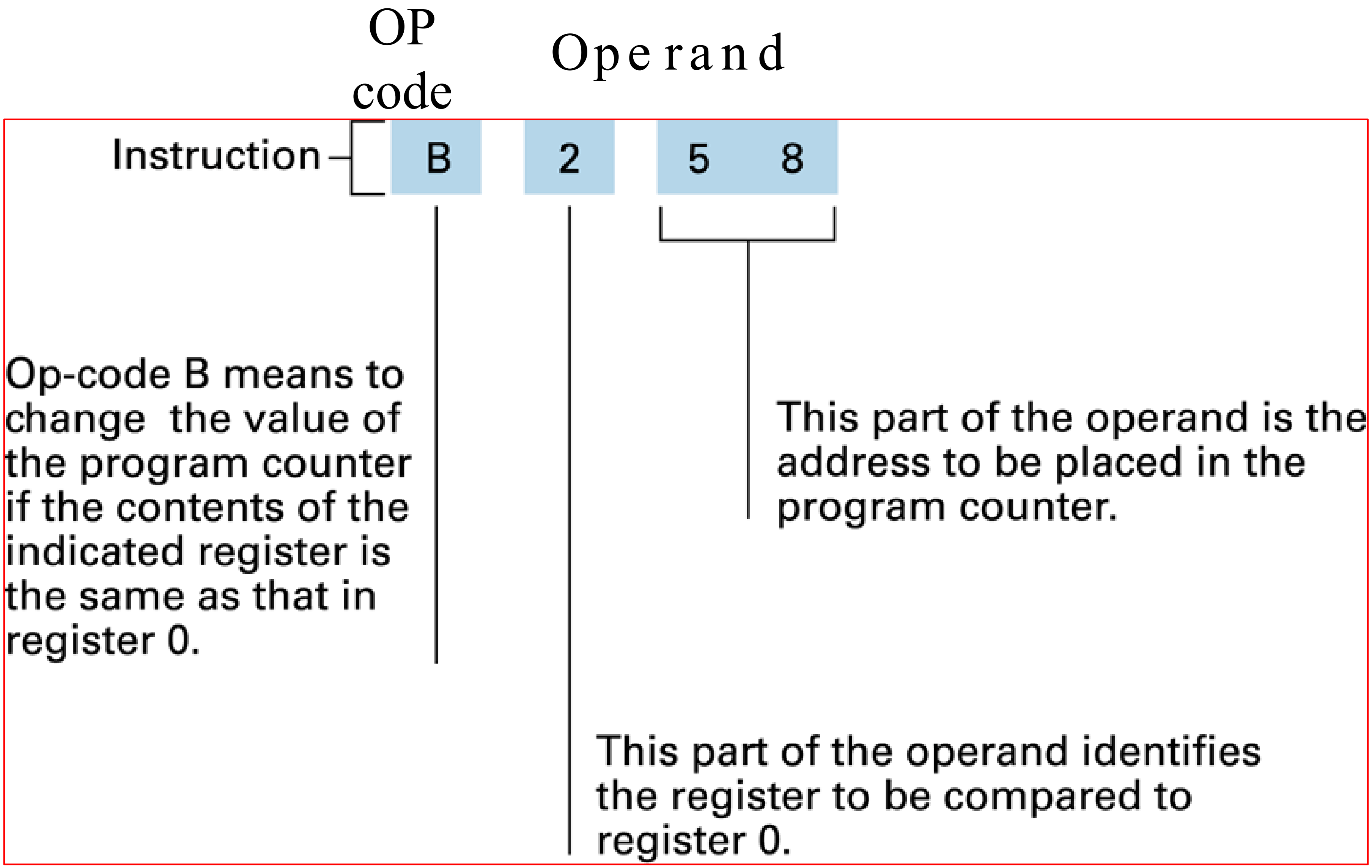
Decode bit pattern in instruction register

3. **Execute**

Perform action requested by instruction in instruction register

Machine cycle

Figure 2.9: Decoding the instruction B258

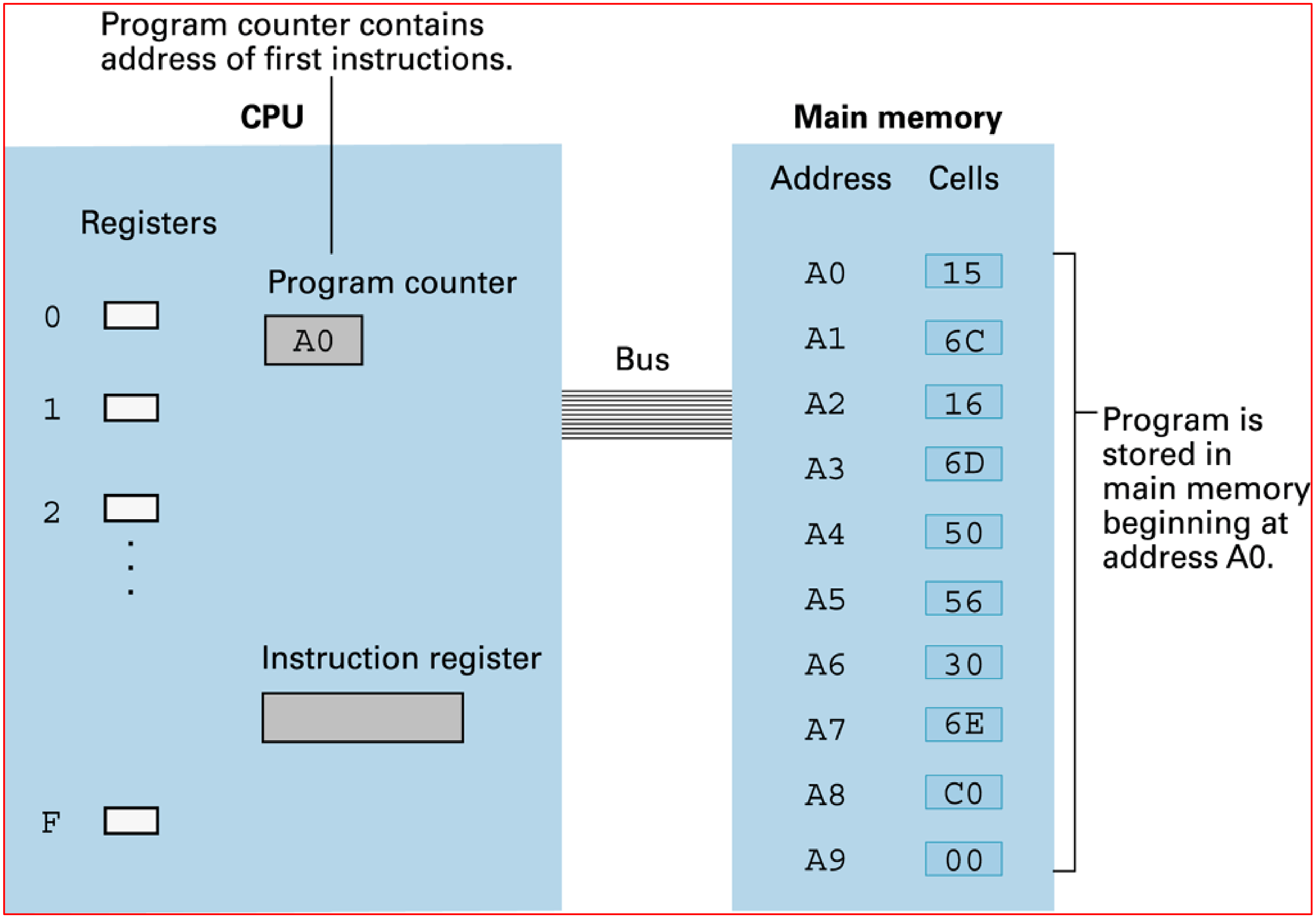


Op-code B means to change the value of the program counter if the contents of the indicated register is the same as that in register 0.

This part of the operand is the address to be placed in the program counter.

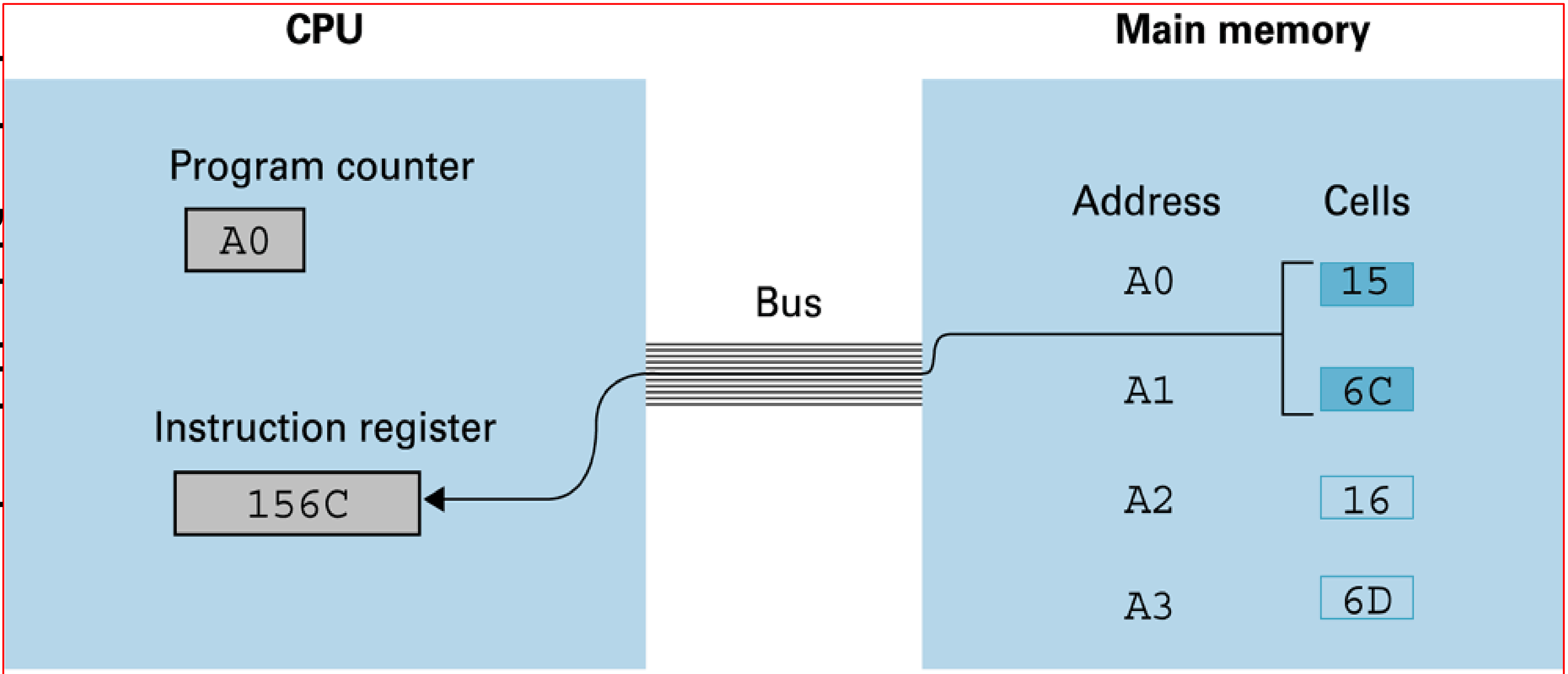
This part of the operand identifies the register to be compared to register 0.

Figure 2.10: The program from Figure 2.7 stored in main memory ready for execution



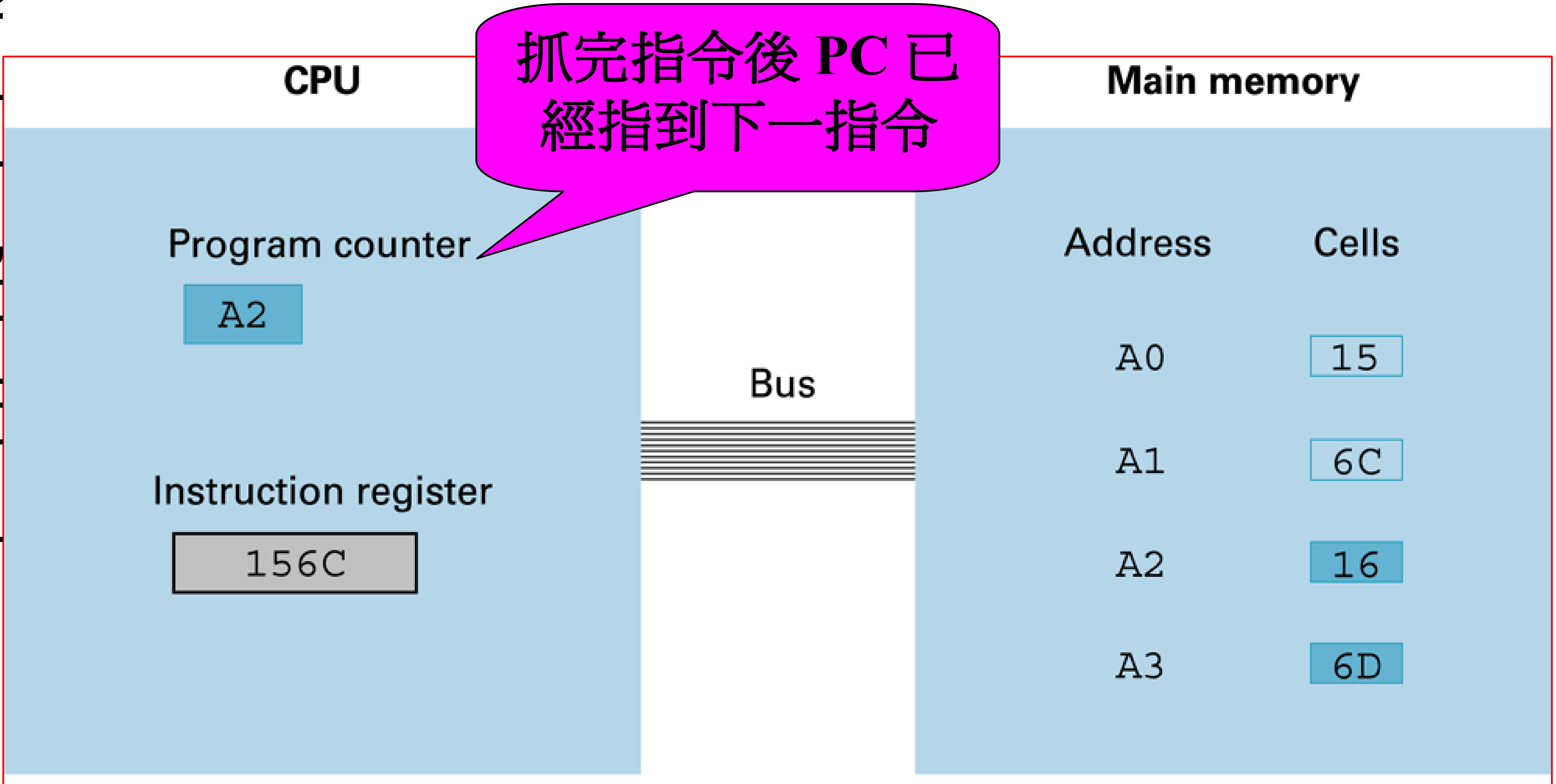
Address	Cells
A0	15
A1	6C
A2	16
A3	6D
A4	50
A5	56
A6	30
A7	6E
A8	C0
A9	00

Figure 2.11a: Performing the fetch step of the machine cycle (continued)



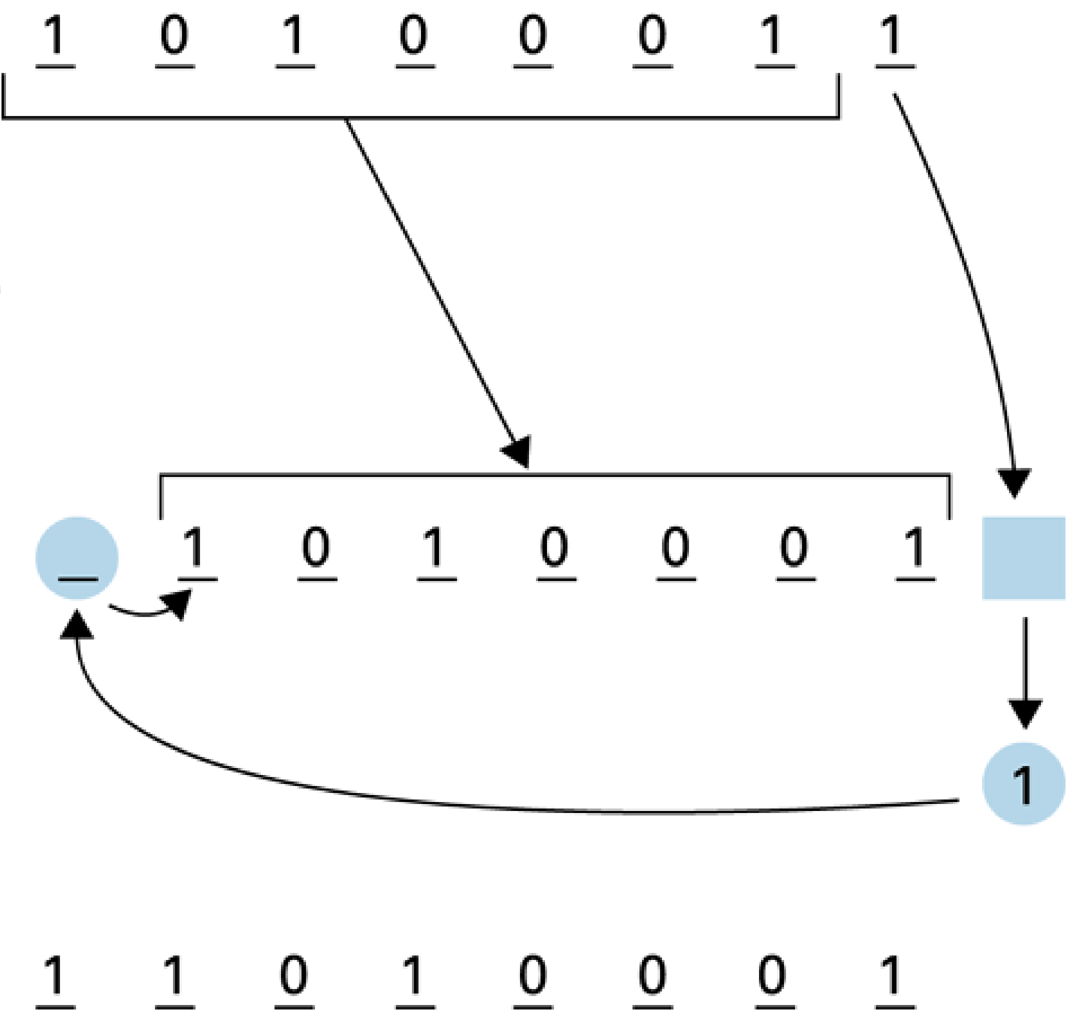
a. At the beginning of the fetch step the instruction starting at address A0 is retrieved from memory and placed in the instruction register.

Figure 2.11b: Performing the fetch step of the machine cycle



b. Then the program counter is incremented so that it points to the next instruction.

Figure 2.12: Rotating the bit pattern A3 one bit to the right



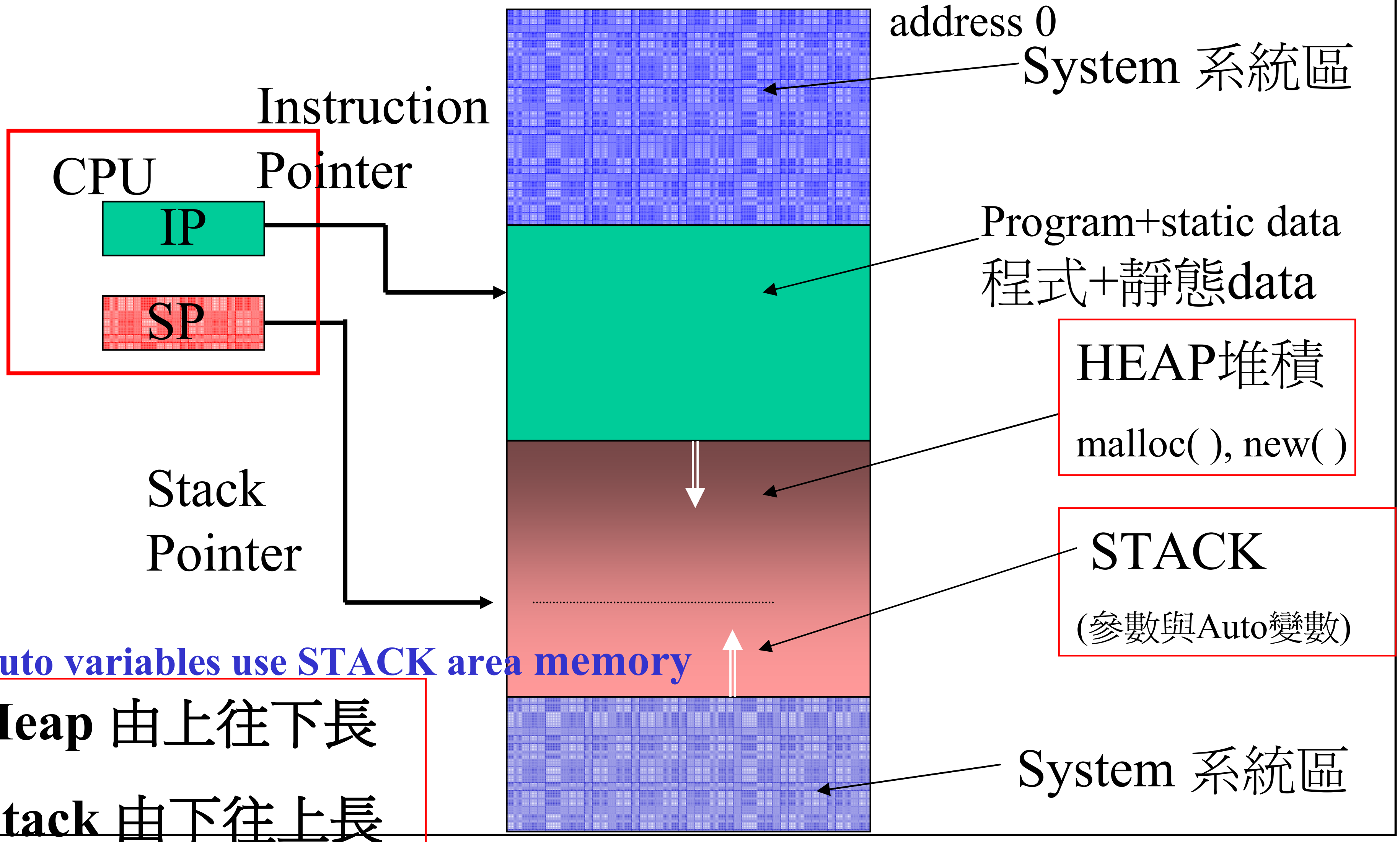
The bit pattern represented by A3 (hexadecimal)

The bits move one position to the right. The rightmost bit “falls off” and is placed in the hole at the other end.

The final bit pattern, which is represented by D1 (hexadecimal)

電腦如何運作?

- **Auto 變數就是沒寫 static 的 Local 變數**



address 0

System 系統區

Instruction Pointer

CPU

IP

SP

Program+static data

程式+靜態data

HEAP堆積

malloc( ), new( )

Stack Pointer

STACK

(參數與Auto變數)

Auto variables use STACK area memory

System 系統區

Heap 由上往下長

Stack 由下往上長

Example of program execution (1/7)

Example of adding 2 values

1. Get first value from memory and place in a register R1
2. Get second value from memory and place in another register R2
3. Activate addition circuitry with R1 and R2 as inputs and R3 to hold results
4. Store contents of R3 (result) in memory
5. Stop

```
1. 156C   LOAD 5, 6C
2. 166D   LOAD 6, 6D
3. 5056   ADD 0, 5, 6
4. 306E   STORE 0, 6E
5. C000   HALT
```

•Each instruction has 2 bytes

•Values to be added stored in 2’s complement notation at memory address 6C and 6D

•Sum placed in memory at address 6E

PC 每次要加 2 因爲每個 instruction 是 2 bytes long

Example of program execution (2/7)

Program stored in Memory

Program

1. 156C
2. 166D
3. 5056
4. 306E
5. C000

Address	Contents
A0	15
A1	6C
A2	16
A3	6D
A4	50
A5	56
A6	30
A7	6E
A8	C0
A9	00

1 memory cell is 8 bits

Program stored in consecutive addresses at address A0

Program Counter PC

Instruction Register **IR**

Program Counter 就是前面說過的 Instruction Pointer

PC 每次要加 2 因爲每個 instruction 是 2 bytes long

Example of program execution (3/7)

Place address **A0** in **program counter** and start machine

Machine Cycle 1

Program Counter PC

Instruction Register **IR**

Fetch	Decode	Execute
•Read from memory instruction at address **A0** •Place instruction 156C in instruction register •Update program counter to be A2 (Why?) •At end of fetch cycle PC: **A2** IR: **156C**	Analyze instruction in IR and deduce need to load R5 with contents of memory cell at address 6C	Load contents of memory cell at address 6C in R5

Control unit starts a new cycle

Example of program execution (4/7)

Machine Cycle 2

PC is **A2**

Program Counter PC

Instruction Register IR

Fetch	Decode	Execute
•Read from memory instruction at address **A2** •Place instruction 166D in instruction register •Update program counter to be A4 •At end of fetch cycle PC: **A4** IR: 166D	Analyze instruction in IR and deduce need to load R6 with contents of memory cell at address 6D	Load contents of memory cell at address 6D in R6

Control unit starts a new cycle

Example of program execution (5/7)

Machine Cycle 3

PC is **A4**

Program Counter PC

Instruction Register IR

Fetch	Decode	Execute
•Read from memory instruction at address **A4** •Place instruction 5056 in instruction register •Update program counter to be A6 •At end of fetch cycle PC: **A6** IR: 5056	Analyze instruction in IR and deduce need to add contents of registers R5 and R6 and place result in R0	Activate 2's complement addition circuitry with inputs R5 and R6 ALU performs addition leaving result in R0

Control unit starts a new cycle

Example of program execution (6/7)

Machine Cycle 4

PC is **A6**

Program Counter PC

Instruction Register IR

Fetch	Decode	Execute
•Read from memory instruction at address **A6** •Place instruction 306E in instruction register •Update program counter to be A8 •At end of fetch cycle PC: **A8** IR: 306E	Analyze instruction in IR and deduce need to store contents of R0 in memory location 6E	Store contents of R0 in memory location 6E

Control unit starts a new cycle

Example of program execution (7/7)

Machine Cycle 5

PC is **A8**

Program Counter PC

Instruction Register IR

Fetch	Decode	Execute
•Read from memory instruction at address **A8** •Place instruction C000 in instruction register •Update program counter to be AA (Why?) •At end of fetch cycle PC: **AA** IR: C000	Analyze instruction in IR and deduce this is a HALT instruction	Machine stops and program is completed

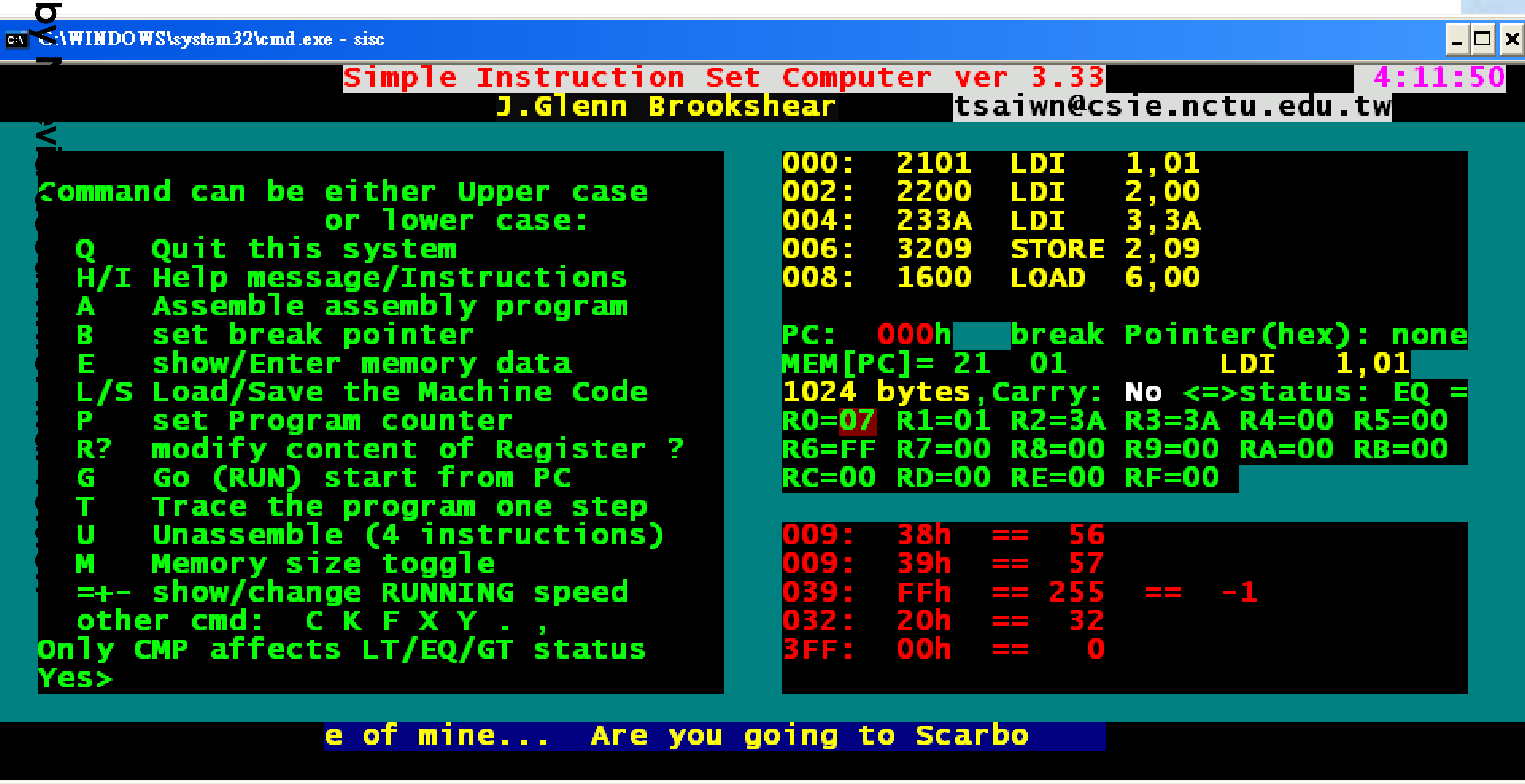
Control unit stops at A8, while the PC is AA; why?

An **Emulator** for this simple instruction computer

- **http://www.csie.nctu.edu.tw/~tsaiwn/sisc/**
  - Download the sisc.zip in that directory
  - Unzip the sisc.zip and you will find a subdirectory named SISC
  - Execute the SISC.EXE in that directory
  - Press ALT_ENTER to get into FULL screen mode
  - You can press H for Help

- sisc.pas, sisc120.c (text mode)

SISC Emulator is running



```
                  Simple Instruction Set Computer ver 3.33          4:11:50
                          J.Glenn Brookshear      tsaiwn@csie.nctu.edu.tw
```

```
Command can be either Upper case
                 or lower case:
  Q   Quit this system
  H/I Help message/Instructions
  A   Assemble assembly program
  B   set break pointer
  E   show/Enter memory data
  L/S Load/Save the Machine Code
  P   set Program counter
  R?  modify content of Register ?
  G   Go (RUN) start from PC
  T   Trace the program one step
  U   Unassemble (4 instructions)
  M   Memory size toggle
  =+- show/change RUNNING speed
  other cmd:  C K F X Y . ,
Only CMP affects LT/EQ/GT status
Yes>
```

```
000:  2101  LDI    1,01
002:  2200  LDI    2,00
004:  233A  LDI    3,3A
006:  3209  STORE  2,09
008:  1600  LOAD   6,00

PC:  000h   break Pointer(hex): none
MEM[PC]= 21  01          LDI    1,01
1024 bytes,Carry: No <=>status: EQ =
R0=07 R1=01 R2=3A R3=3A R4=00 R5=00
R6=FF R7=00 R8=00 R9=00 RA=00 RB=00
RC=00 RD=00 RE=00 RF=00

009:  38h   ==  56
009:  39h   ==  57
039:  FFh   == 255   ==  -1
032:  20h   ==  32
3FF:  00h   ==   0
```

```
                 e of mine...  Are you going to Scarbo
```

SISC extension Instructions

- In addition to instructions on our text book ("?" means don't care)
- **D0??/D1?? Get/Put Char to/from R0**
- **D2??/D3?? Get/Put int to/from R0**
- **D5XY/D6XY In/Put string to/from XY**
- **EZXY (zzzz xxxx yyyy in binary):**
- **Ezz00 xxxx yyyy: LOAD F,zzxxxxyyyy**
- **Ezz01 xxxx yyyy:STORE F,zzxxxxyyyy**
- **Ezz10 xxxx yyyy: CALL zzxxxxyyyy**
- **E??11 ???? ????: RETurn**
- **FZXY compare/conditional Jump**
- **zzzz in binary:**
- **??00 xxxx yyyy: CMP RX to RY**
- **zz01 xxxx yyyy: JLT zzxxxxyyyy**
- **zz10 xxxx yyyy: JEQ zzxxxxyyyy**
- **zz11 xxxx yyyy: JGT zzxxxxyyyy**
- **Only CMP affects LT/EQ/GT status**

A SISC example (Assembly program) to print its content in decimal samp.asm (1/2)

```
 ORG 0
 LDI 1, 1 ;   2 1,01  ; R1=1  (00)
 LDI 2, 0 ;   2200   ; R2=0   (02)
 LDI 3, 58 ;  233A   ; R3 = length of this program
AGAIN:  STORE 2, THERE+1 ; 3209   ; store r2 into 9 ; STORE 2, 9
THERE: LOAD 6,0  ;  16 00  ; LOAD r6 from ?? (:08 :09 )
 LDI 0,'M' ;  20 4d  ; r0="M"  或寫成 LDI 0, 77  LDI 0,$4d
 PUTC      ;  d1 00  ; print "M"
 LDI 0, '(' ; 20 28  ; "(" ===    LDI 0,40  或 LDI 0,$28 或 LDI 0,28h
 PUTC      ;   d100   ; print "("   (:10h)
 MOV  2,0 ;   40 20   ; move r2 to r0  (:12h)
 PUTI     ;   d300   ; print r0 as integer
 LDI 0, ') ';  20 29 ; ") "   === LDI 0, 41 或  LDI 0,29h
 PUTC      ;  d1 00
 LDI 0, '=' ; 203d;"="      === LDI 0, 61 或  LDI 0,2dh
 PUTC      ;  d1 00
```

A SISC example (Assembly program) to print its content in decimal samp.asm (2/2)

```
 MOV  6,0  ;  40 60   ; move r6 to r0
 PUTI      ;  d300   ; print r0 as integer
 LDI  0,13 ;  200d; CR
 PUTC      ;  d1 00
 LDI  0,10 ;  20 0A ;LF
 PUTC      ;  d1 00
 MOV  3,0  ; 40 30 ; move r3 to r0 (program length)
 ADD 2,2,1 ; 5221  ; r2 := r2+1 (r1 contains 1)
 BR 2,OK   ; B232  ; jump to OK if R2=R0=R3
 BR  0,AGAIN ; B006  ; goto :AGAIN=06
OK:  LDI 0,7   ; 2007  ; bell
 PUTC      ; d1 00
 HALT      ; c000  ; halt
 END
```

The machine code for the samp.asm

samp.asm → samp.mc

```
;分號開始或空白行都會被忽略
;空白和逗號可出現於任何地方
2 1,01  ; R1=1  (00)
2200   ; R2=0   (02)
233A   ; R3 = length of this program
3209   ; store r2 into 9 (:AGAIN = 06)
16 00  ; LOAD r6 from ?? (:08 :09 )
20 4d  ; r0="M"
d1 00  ; print "M"
20 28  ; "("

d100   ; print "("   (:10h)
40 2,0   ; move r2 to r0  (:12h)
d300   ; print r0 as integer
20 29 ; ")"
d100
203d;"="
d100
203d;"="
d100
```

```
4060; move r6 to r0
d300 ; out content of mem[r2] (:20h)
200d; CR
d100
20 0A ;LF
d100  ; print Line Feed   (:28h)
40 30 ; move r3 to r0 (program length)
5221  ; r2 := r2+1 (r1 contains 1)
B232  ; jump to done if R2=R0=R3
b006  ; goto :AGAIN=06  (this line :30)
2007  ; bell (:done = 32)
d100  ; beep the speaker
c000  ; halt
ffff  ; for man check only
```

Figure 2.13: Controllers attached to a machine's bus

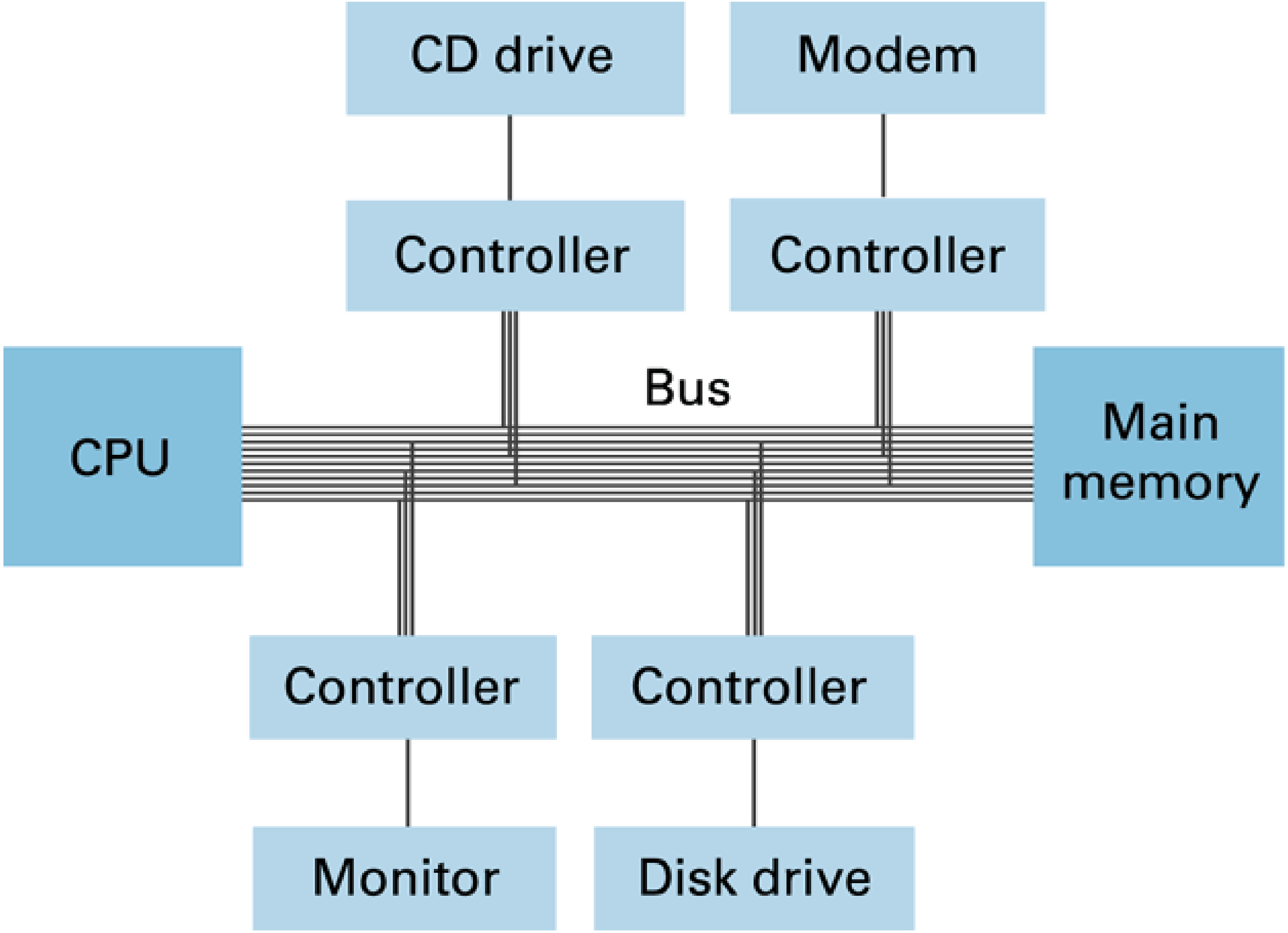
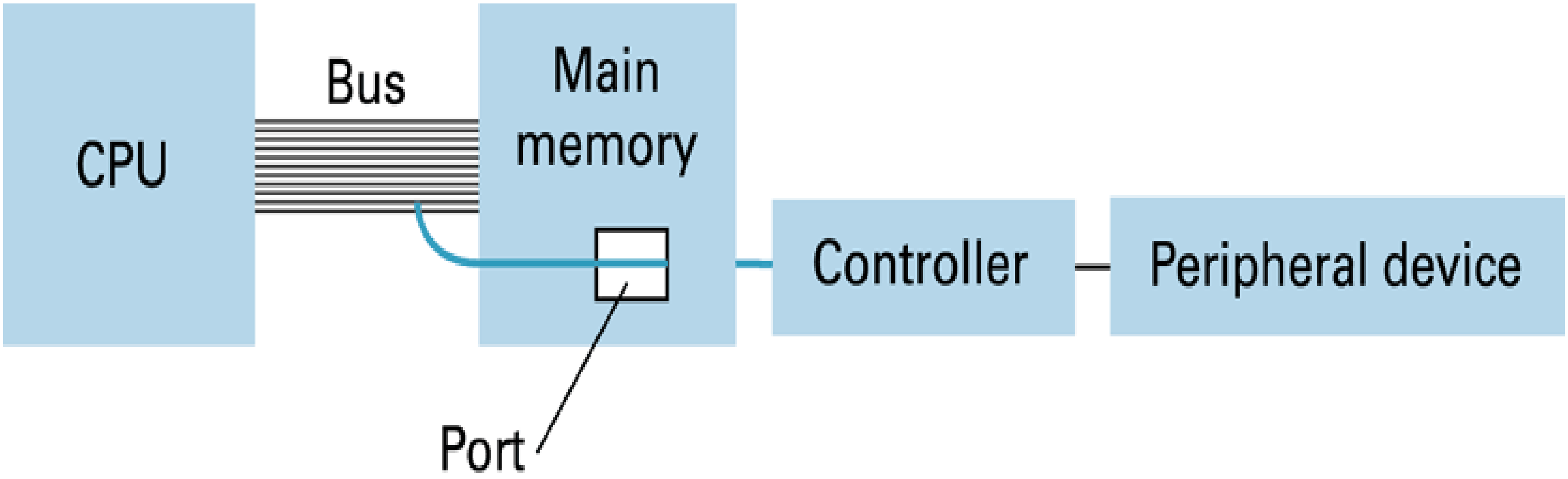


Figure 2.14: A conceptual representation of memory-mapped I/O



Direct memory access

- **Direct memory access (DMA)** is a process in which an external device takes over the control of system **bus** from the CPU.
- DMA is for **high-speed data transfer** from/to mass storage peripherals, e.g. Hard Disk drive, magnetic tape, CD-ROM, and sometimes video controllers.
- A DMA controller interfaces with several peripherals that may request DMA.
- The DMA controller handles these data transfers between the main memory and the interface controllers bypassing the CPU.
- The basic idea of **DMA** is to transfer *blocks of data* directly between memory and peripherals. The data don't go through the microprocessor but the data bus is occupied.
- "Normal" transfer of one data byte takes up to 29 clock cycles. The DMA transfer requires only 5 clock cycles.

Other Machine Architectures

CISC Complex Instruction Set Computer

- Complex machine that can decode and execute a wide variety of instructions
- Easier to program (single instruction performs the task of several instructions in RISC)
- Complex CPU design
- To reduce required circuitry, use *microprogram* approach where each machine instruction is actually executed as a sequence of simpler instructions
- Example is Pentium processors by Intel

RISC Reduced Instruction Set Computer

- Simple machine that has a limited instruction set
- Simpler CPU design
- Machine language programs are longer than CISC counterpart because several instructions are required to perform the task of a single instruction in CISC
- Example is PowerPC developed by Apply, IBM and Motorola

- **Other concepts: Pipelining, multiprocessor machines**

Unpipelined Microprocessors

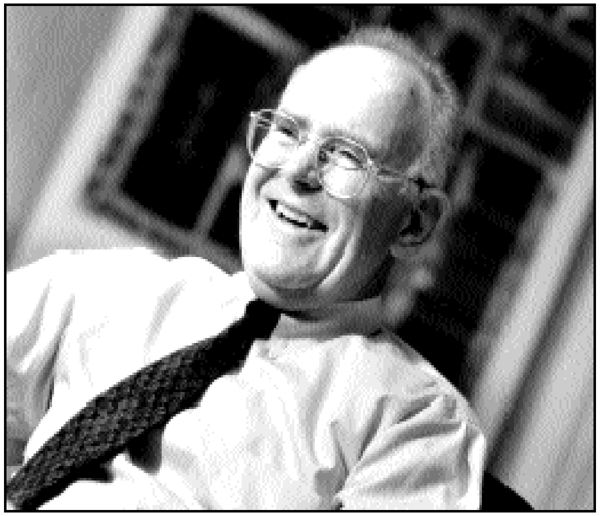
- Typically an instruction enjoys **five phases** in its life
  - Instruction **fetch** from memory
  - Instruction **decode** and **operand fetch**
  - Execute
  - Data memory access
  - Register write
- Unpipelined execution would take a long single cycle or multiple short cycles
  - **Only one instruction inside processor at any point in time**

Pipelining (管線; 流水線)

- One simple observation
  - Exactly **one piece of hardware is active** at any point in time
- Why not fetch a new instruction every cycle?
  - **Five instructions in five different phases**
  - **Throughput** increases five times (ideally)
- Bottom-line is
  - If consecutive instructions are independent, they can be processed in parallel
  - The first form of Instruction-Level Parallelism (**ILP**)

Moore's Law (摩爾定律)

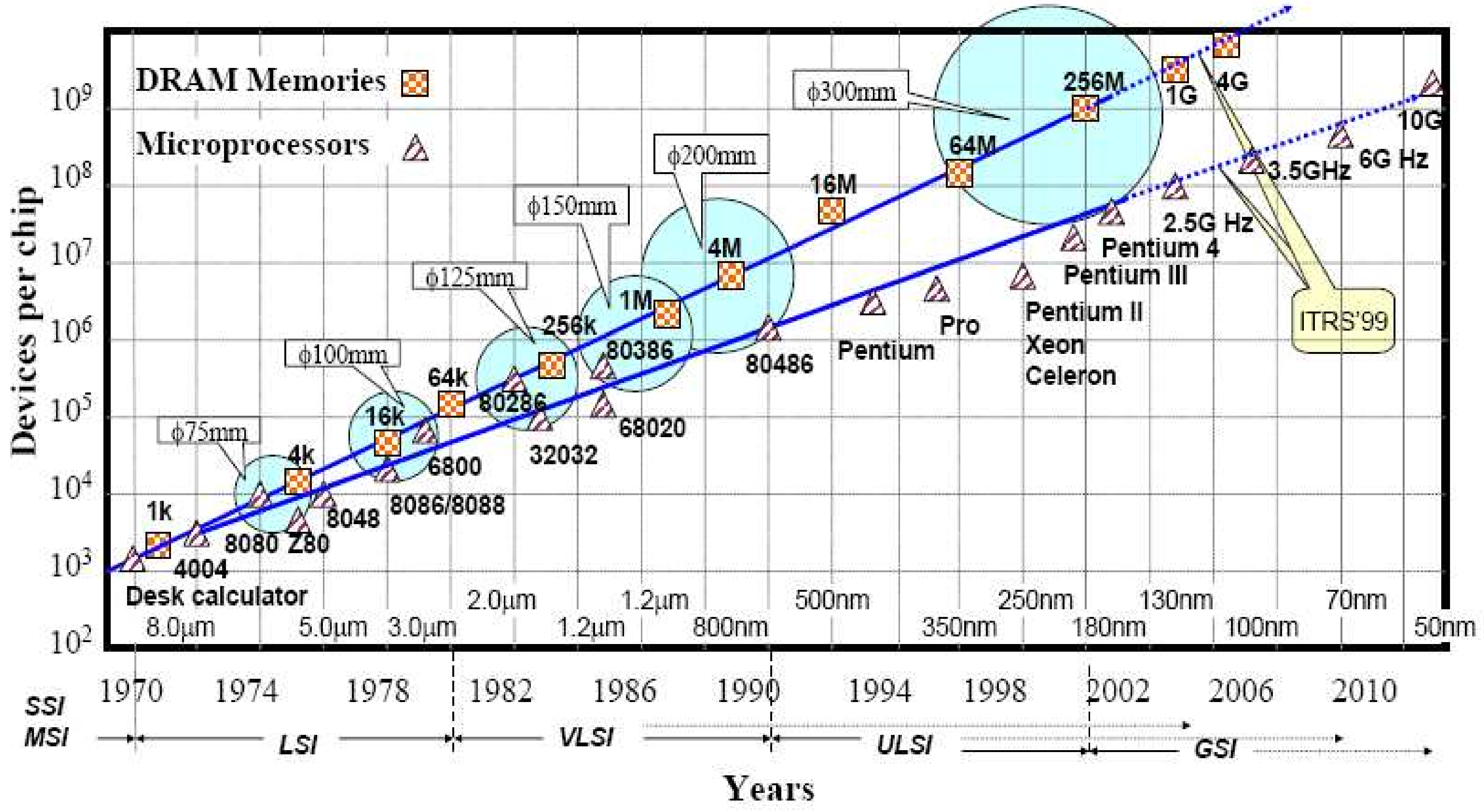
- Number of transistors on-chip doubles every 18 months
  - So much of innovation was possible only because we had **transistors**
  - Phenomenal 58% performance growth every year
- Moore's Law is facing a danger today
  - **Power consumption** is **too high** when clocked at multi-GHz frequency and it is proportional to the number of switching transistors
- Wire delay doesn't decrease with transistor size



Gordon Moore (co-founder of Intel) predicted in 1965 that the transistor density of semiconductor chips would double roughly every 18 months.

Murphy's Law

Moore's Law and Technology Scaling



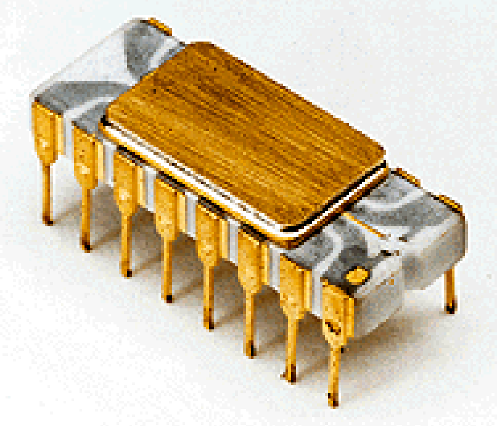
…the performance of an IC, including the number components on it, doubles every 18-24 months with the same chip price ... - Gordon Moore - 1960

http://www.acm.vt.edu/~andrius/work/microproc/

- 1970 Intel releases DRAM memory chip
- Intel 4004 (1971)

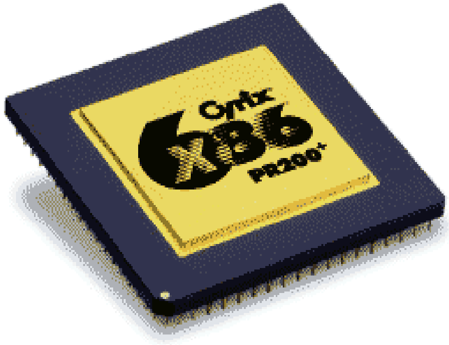
  1MHz, 45 instructions

  = 2300 transistors
- **1972 Intel 8008**
- **1974 Intel 8080, 2-MHz**
- **1974 Motorola 6800**
- **1975 Zilog Z80, 1976 MOS Technologies 6502**
- **1978 Intel 8086, 4.77MHz, 29000 transistors**
- **1979 Intel 8088**
- **1982 Intel286, 12 MHz**
- **1985 Intel386 , first 32-bit, 25MhZ**



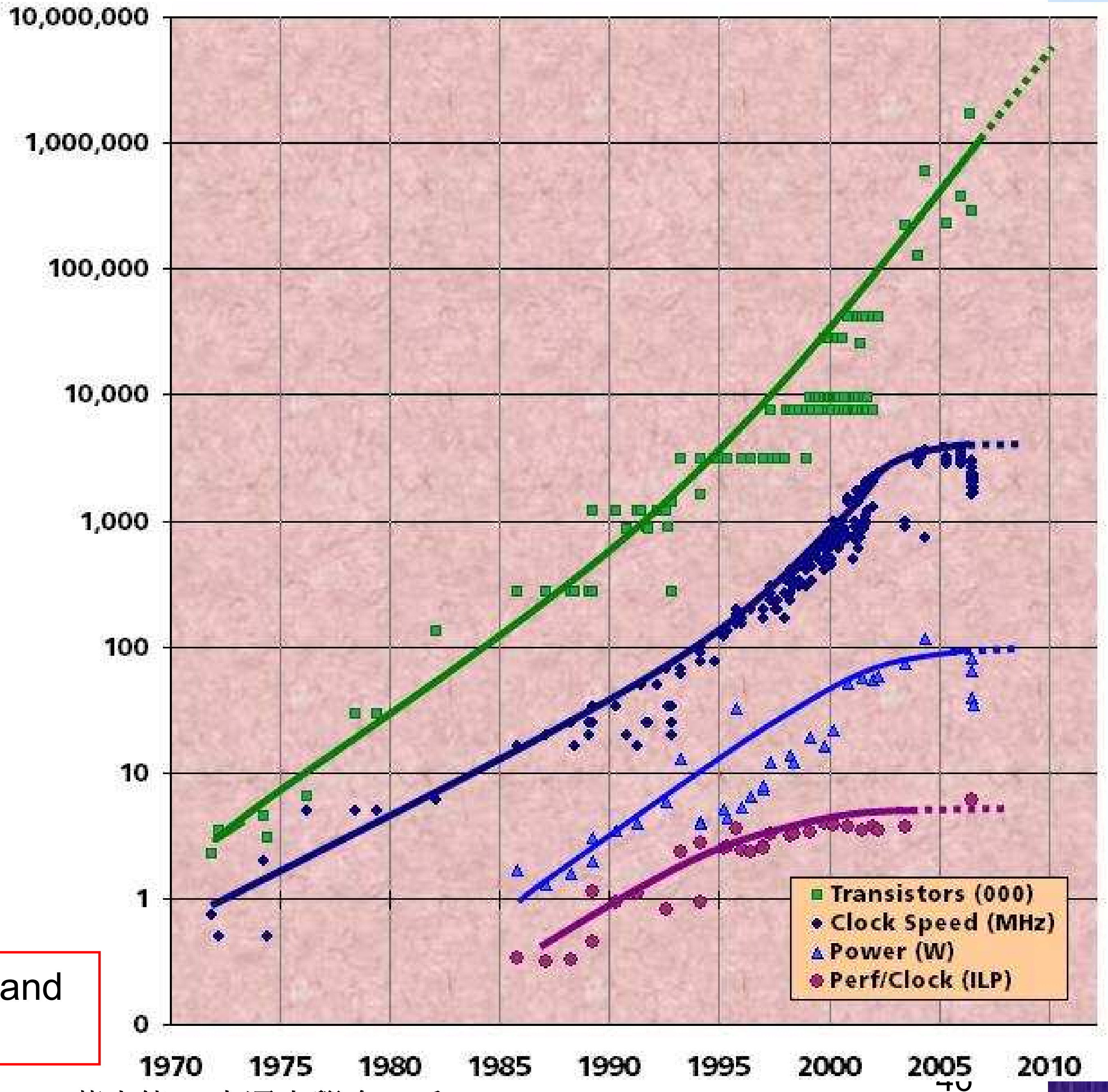
http://www.pcmech.com/show/processors/35/2/

- 1989 Intel 486 DX (with 487), 33 MHz
- 1994 AMD 486
- 1993 Intel Pentium, 60MHz
- 1995 AMD AM5x86, 133MHz
- 1995 Intel Pentium Pro
- 1995 Cyrix 6x86
- 1996 AMD K5
- 1997 Pentium MMX
- 1997 Pentium II, 1998 Celeron, 1999 P !!!
- 2000 Celeron II, Pentium IV, AMD Duron
- 2003/03/12 Intel® **Centrino™（迅馳™）**
- **Core 2 Duo, Quad-core, 2008/11/17 Core-i7, 2009/7 core i5, core i3**
- **2010 Core i7-980x (6-core)**



Revolution is Happening Now

- Chip density is continuing increase ~2x every 2 years
  - Clock speed is not
  - Number of processor cores may double instead
- There is little or no more hidden parallelism (ILP) to be found
- Parallelism must be exposed to and managed by software



Source: Intel, Microsoft (Sutter) and Stanford (Olukotun, Hammond)

Multi-core (多核心)

- Put a few reasonably complex processors or many simple processors on the chip
  - Each processor has its own primary cache and pipeline
  - Often a processor is called a **core** (**核心**)
  - Often called a **Chip-MultiProcessor** (**CMP**)
- Did we use the transistors properly?
  - Depends on if you can keep the cores busy
  - Introduces the concept of **Thread-Level** Parallelism (**TLP**)

Concurrent (Parallel) programming

Core memory 磁蕊記憶體

Core Dump 記憶體**傾倒**(存)

Communication in Multi-core

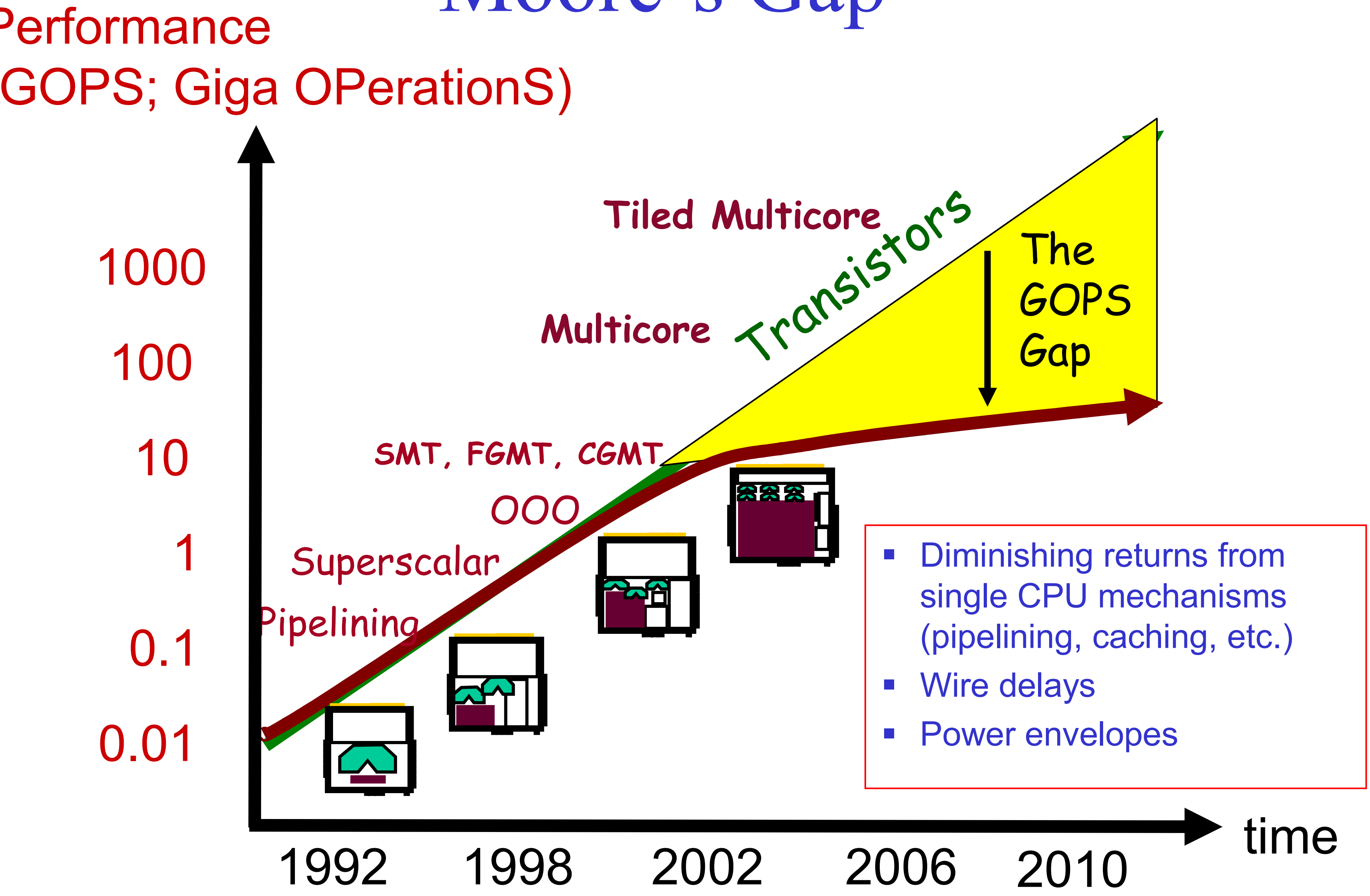
- Ideal for **shared address space**
  - Fast on-chip hardwired communication through cache (no OS intervention)
  - Two types of architectures
    - Private cache CMP: each core has its private cache hierarchy (no cache sharing); Intel Pentium D, Dual Core Opteron, Intel Montecito, Sun UltraSPARC IV, IBM Cell (more specialized)
    - Shared cache CMP: Outermost level of cache hierarchy is shared among cores; **Intel Woodcrest (Server-grade Core duo), Intel Conroe (Core2 duo for desktop)**, Sun Niagara, IBM Power4, IBM Power5

Thread-level Parallelism

- Look for concurrency at a granularity coarser than instructions
  - Put a chunk of consecutive instructions together and call it a thread (largely wrong!)
  - Each thread can be seen as a “dynamic” subgraph of the sequential control-flow graph: take a loop and unroll its graph
  - The edges spanning the subgraphs represent data dependence across threads
    - The goal of parallelization is to minimize such edges
    - Threads should mostly compute independently on different cores; but need to talk once in a while to get things done!

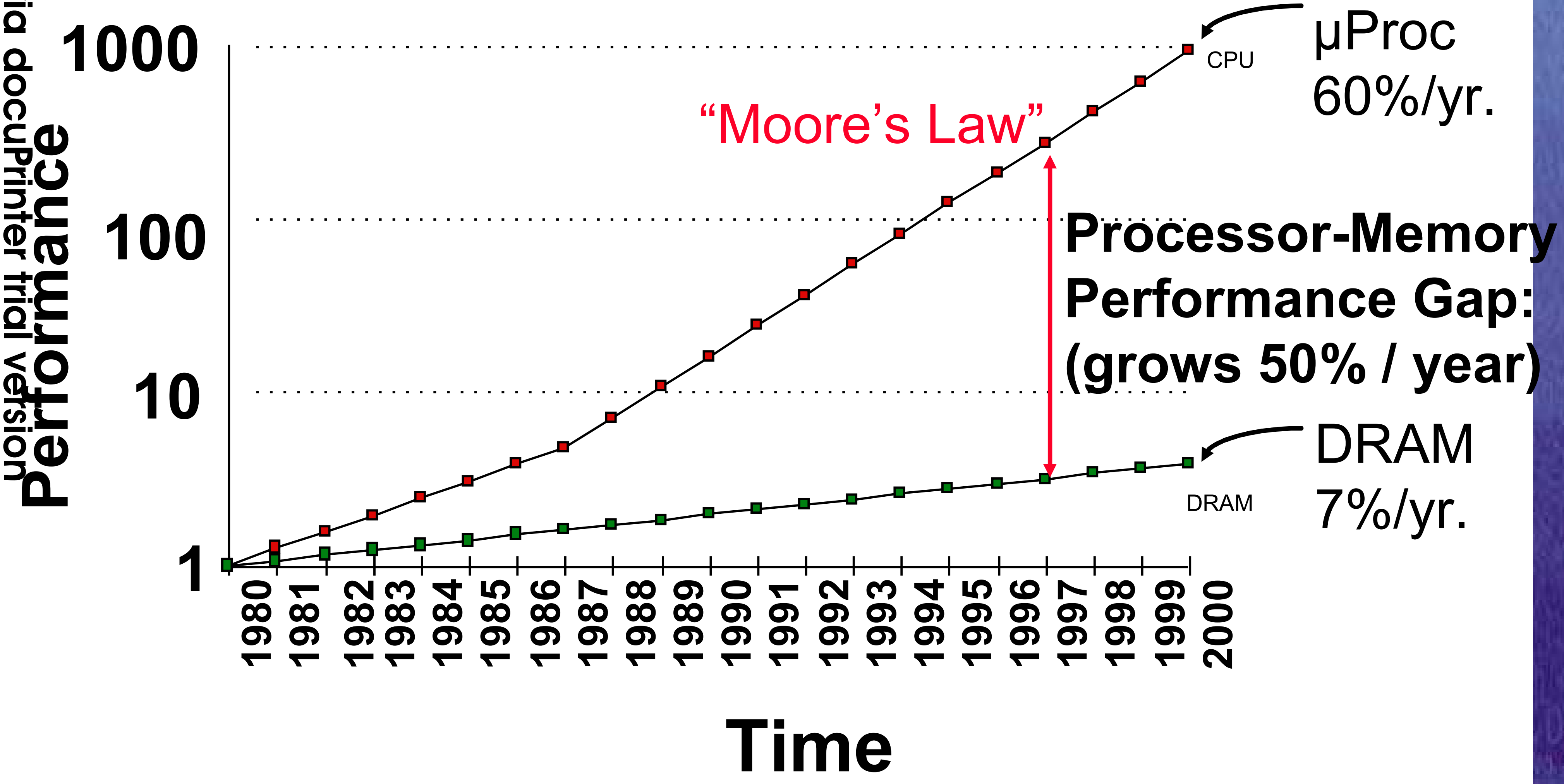
用 Java 練習寫 Thread 很簡單 ☺

"Moore's Gap"

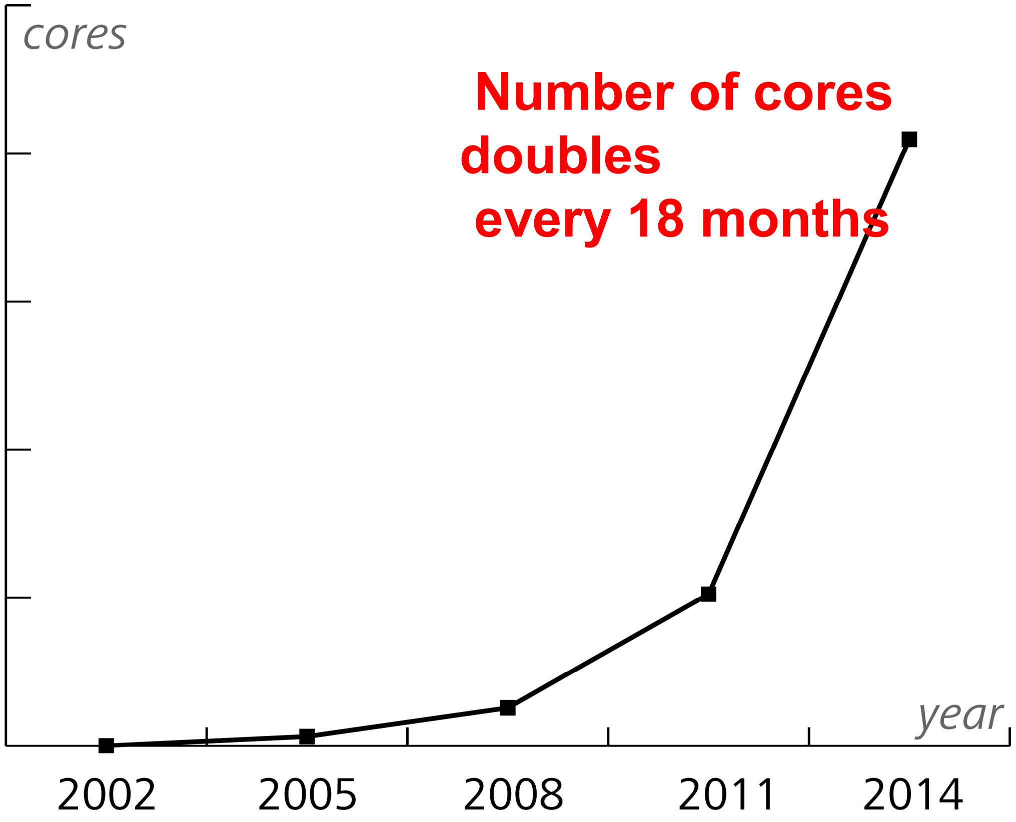


- Diminishing returns from single CPU mechanisms (pipelining, caching, etc.)
- Wire delays
- Power envelopes

Processor-DRAM Gap (latency)



The Future of **Multi_core**



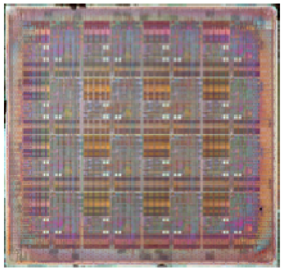
Number of cores doubles every 18 months

Parallelism replaces clock frequency scaling and core complexity

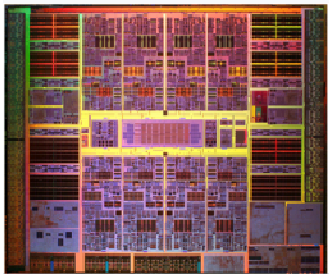
Resulting Challenges…

Scalability
Programming
Power

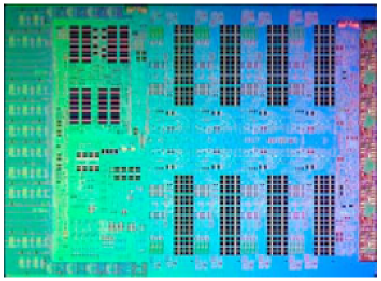
MIT RAW



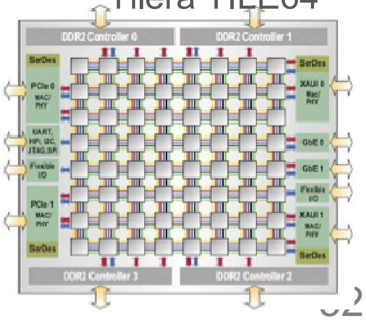
Sun Ultrasparc T2



IBM XCell 8i



Tilera TILE64



Gflop/s 1988, Tflop/s 1997, Pflop/s 2008, *Exaflop/s ~2018 ??*

Flops= Floating OPs

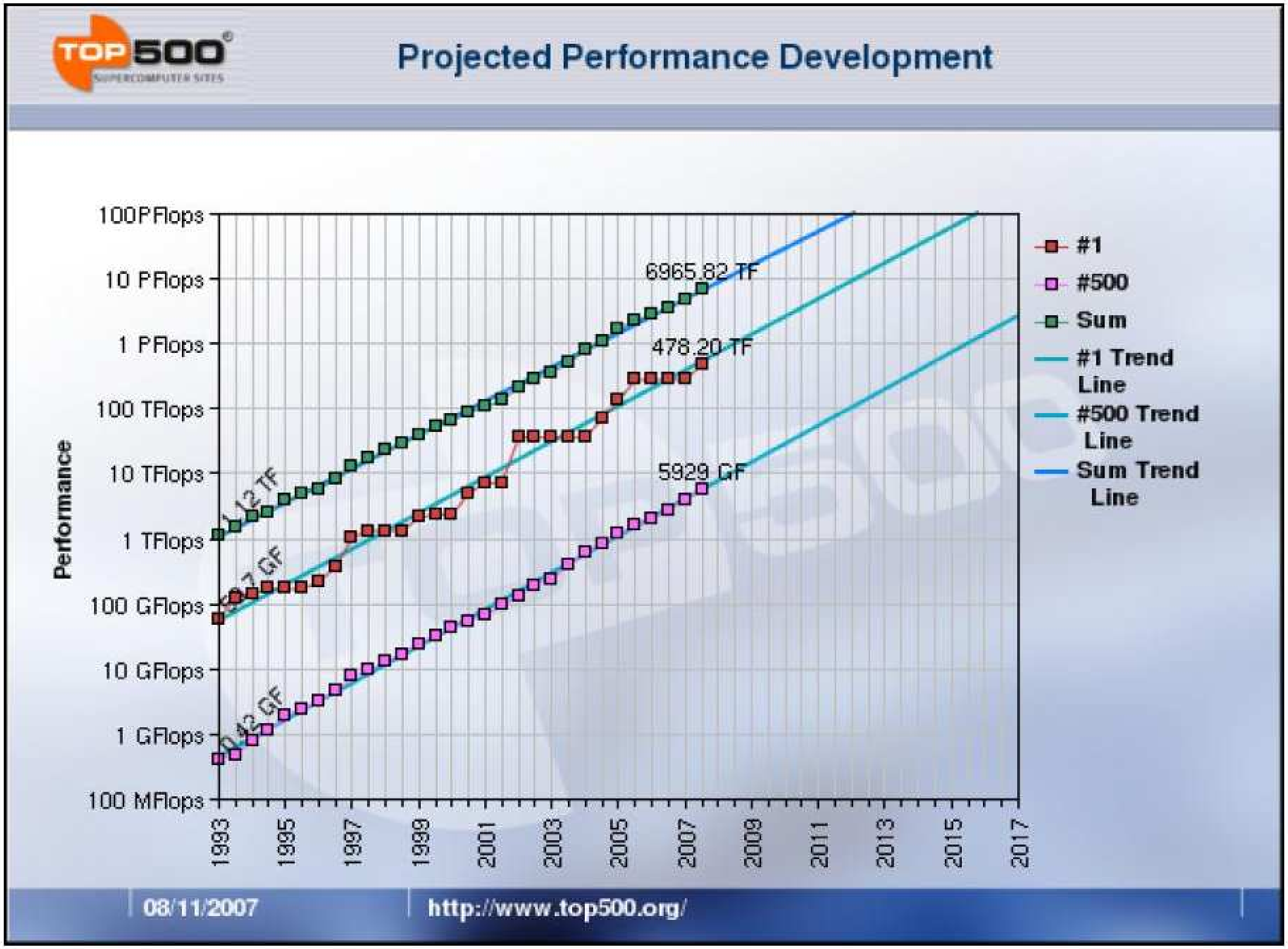
Kilo, Mega, **Giga, Tera, Peta, Exa**, Zeta, Yotta

In the next 10 years technology projections point to enormous raw compute capabilities:

- Performance increases 3 orders of magnitude every 10 years

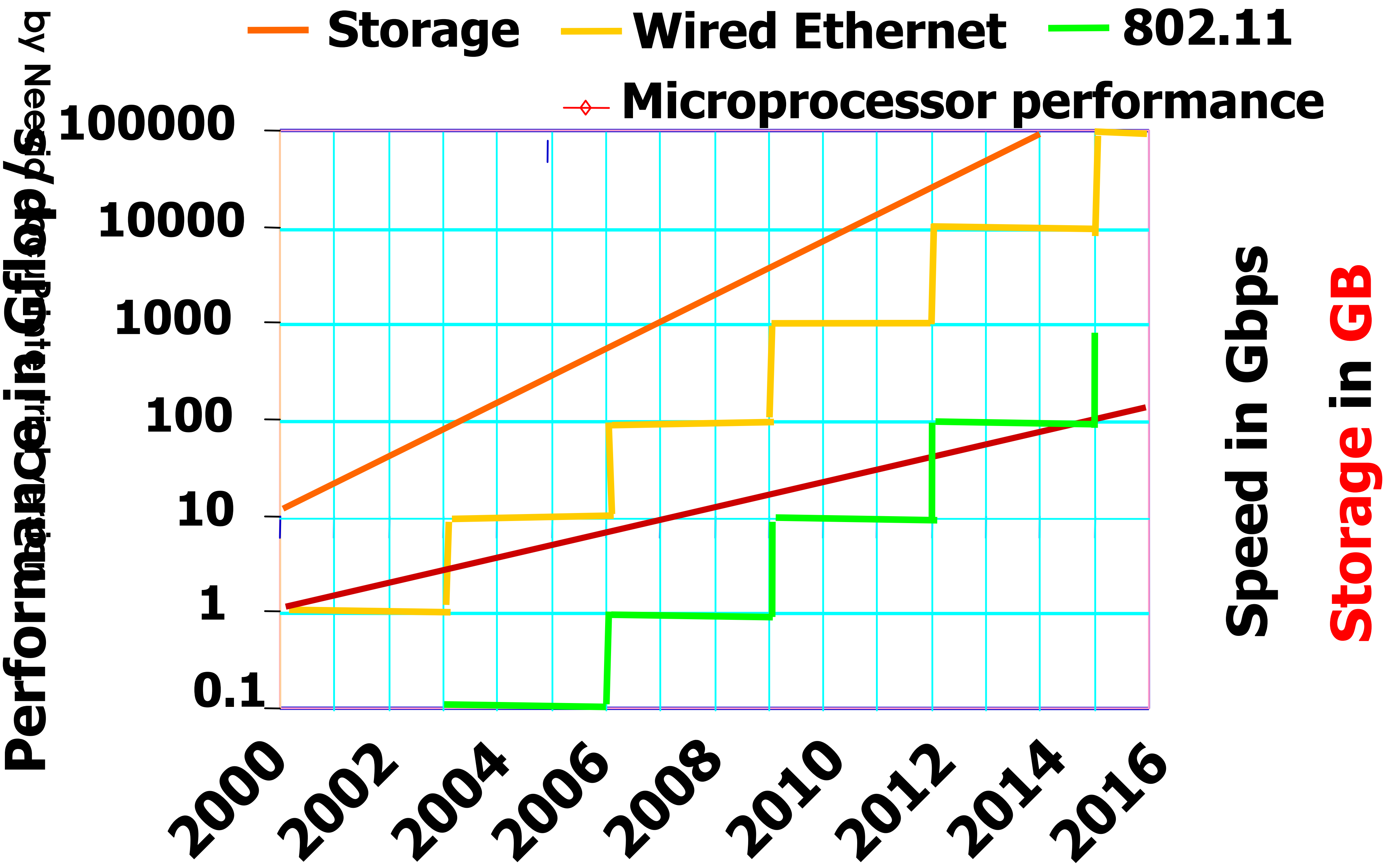
So what?

- Can we realize these raw compute capabilities in systems that can deliver real value? How?
- And, more importantly, what might we do with these capabilities?



c/o Tilak Agarwala (IBM)/adapted by Keyes

Moore's Law vs. Gilder's Law



Measuring Memory Capacity

Km = 千米=公里

- **Kilobyte:** 2^{10} bytes = 1024 bytes
  - Example: 3 KB = 3 times1024 bytes
  - Sometimes “kibi” rather than “kilo”
- **Megabyte:** 2^{20} bytes = 1,048,576 bytes
  - Example: 3 MB = 3 x 1,048,576 bytes
  - Sometimes “megi” rather than “mega”
- **Gigabyte:** 2^{30} bytes = 1,073,741,824 bytes = 10^9 bytes
  - Example: 3 GB = 3 x 1,073,741,824 bytes
  - Sometimes “gigi” rather than “giga”

- Tera = 1024 Giga = 10^9
- Peta = 1024 Tera = 10^{12}
- Exa = 1024 Peta = 10^{18}

- Zeta = 1024 Exa = 10^{21}
- Yotta = 1024 Zeta = 10^{24}

μs = micro second

760 *mm* Hg (Atmospheric pressure)

- ***d*** **= deci = 10^{-1}**
- ***c*** **= centi = 10^{-2}**
- ***m*** **= milli = 10^{-3}**
- ***μ*** **= micro = 10^{-6}**
- ***n*** **= nano = 10^{-9}**
- ***p*** **= pico = 10^{-12}**

- *f* = femto = 10^{-15}
- *a* = atto = 10^{-18}
- *z* = zepto = 10^{-21}
- *y* = yocto = 10^{-24}

cm = 公分=厘米

mm = 毫米

http://en.wikipedia.org/wiki/Exa-

BACK

莫非定律 (Murphy's Law)

- 莫非定律 (Murphy's Law)：

  Anything that can go wrong will go wrong.

  只要會出差錯的事情, 它就會出差錯。

- **Murphy**(莫非)者，查無其人，**是個虛構人物**。1950年代美國海軍的教育宣導卡通裡面，有個笨手笨腳的機械士叫做Murphy。**所謂莫非定律，最早就是出自這部卡通。**
- **排隊時別人排的那排就是比你排的那排快 ;-)**
- **你在電梯內想著會不會停電被關住阿就. . .**

Chapter 2 Data Manipulation

Q&A