# SYST 27198

## CPU Architecture and Assembly

**Dr. Rachel Jiang**

Sheridan Institute of Technologies and Advanced Learning

Winter 2017

# Acknowledgement

This notes is based on Professor Victor Ralevich's course Notes of

"**Structured Computer Organization**"

Updated by Rachel Jiang

**Jan. 2017**

# Chapter Two

## Computer System Organization

"*Hardware and software are logically equivalent.*" - Andrew S.Tanenbaum

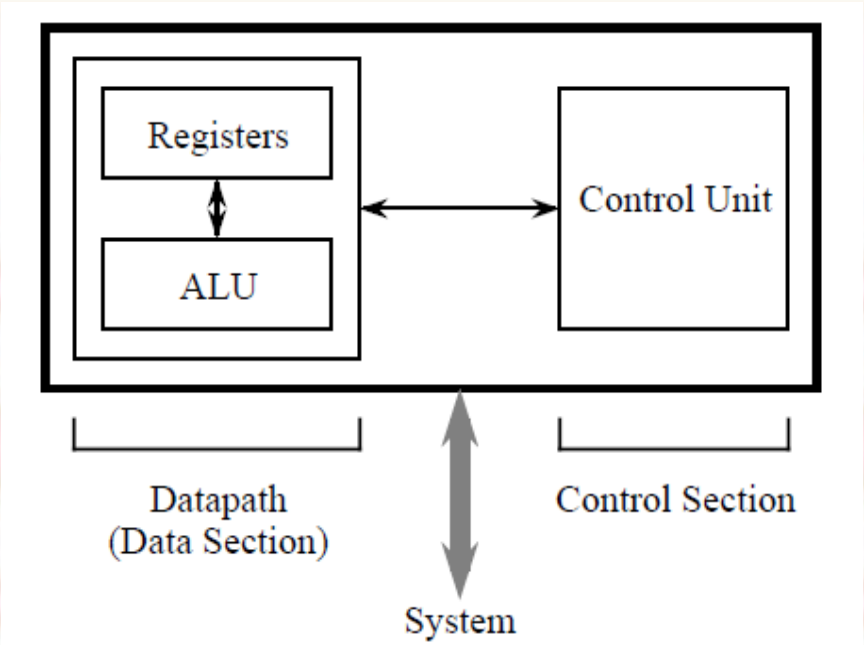
"*Hardware is just petrified software.*" - Karen Panetta Lentz

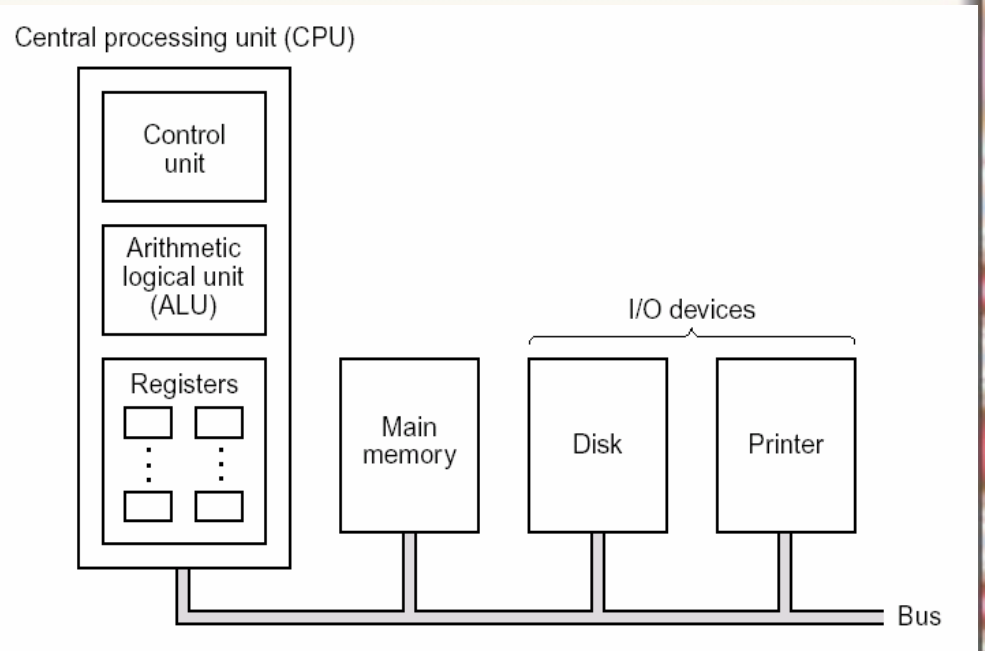
# Topics:

- General CPU Components
- RISC and CISC architectures
- Design principles for modern computers
- Instruction-level parallelism
- Processor-level parallelism
- Primary and secondary memory
- Logical structure of a simple personal computer
- Input/Output
- Performance measurement

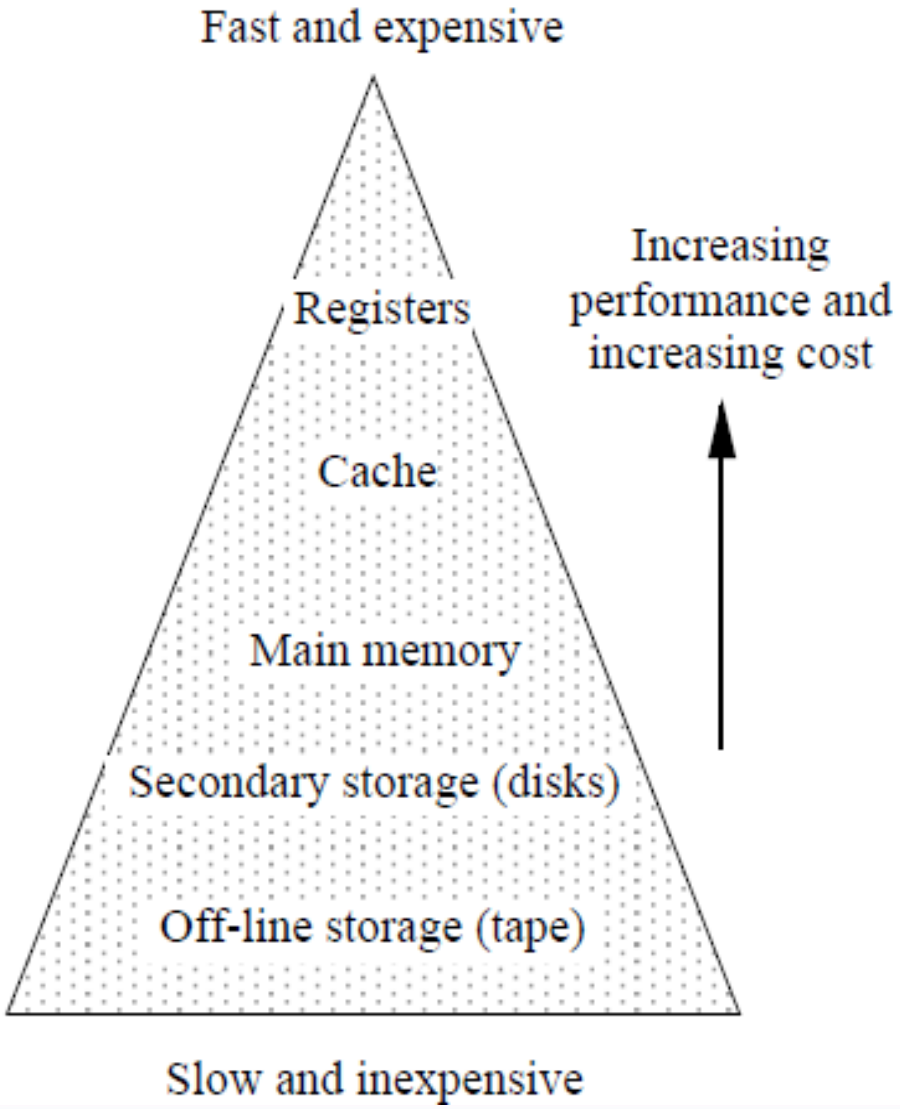
# General CPU Components

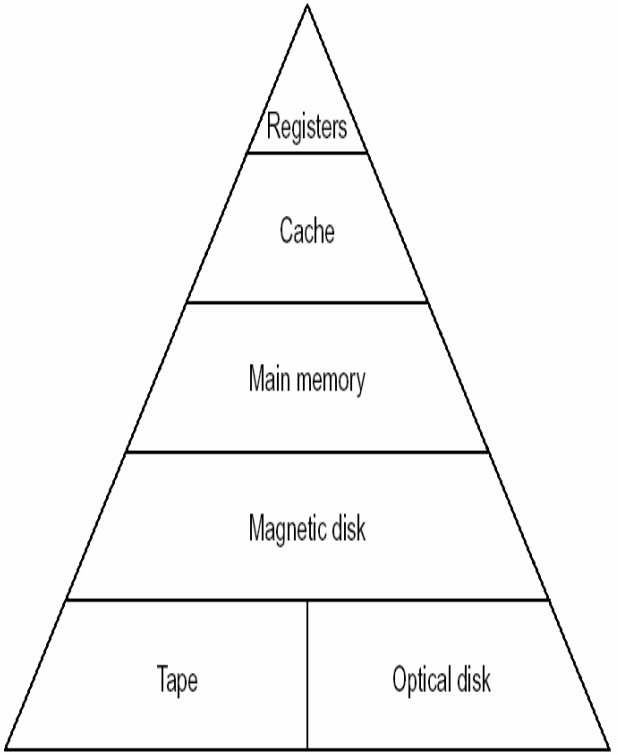
- **CPU** (Central Processing Unit)
  - executes programs stored in the main memory by fetching, examining, and executing their instructions one after another (von Neumann).

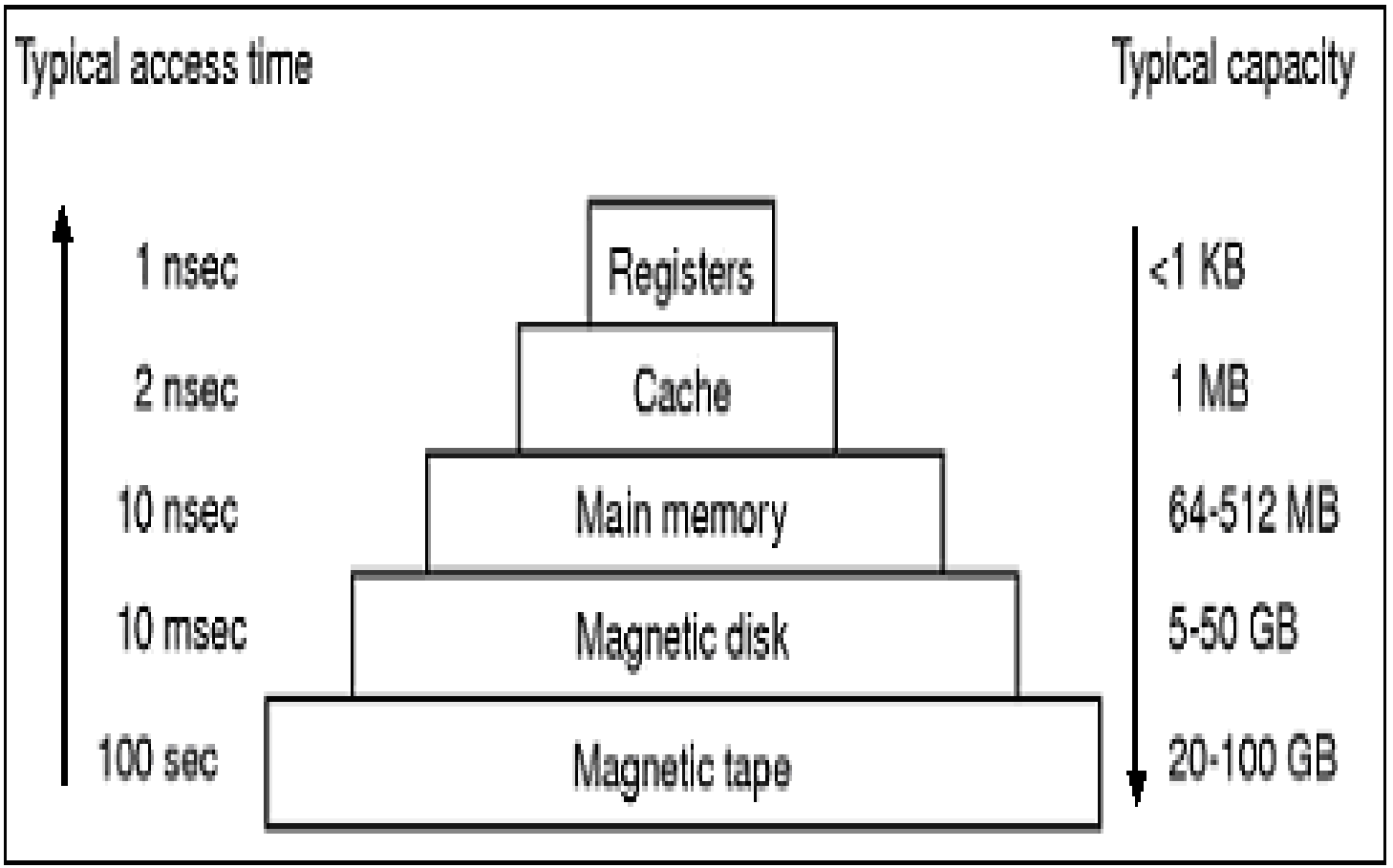


# The Memory Hierarchy



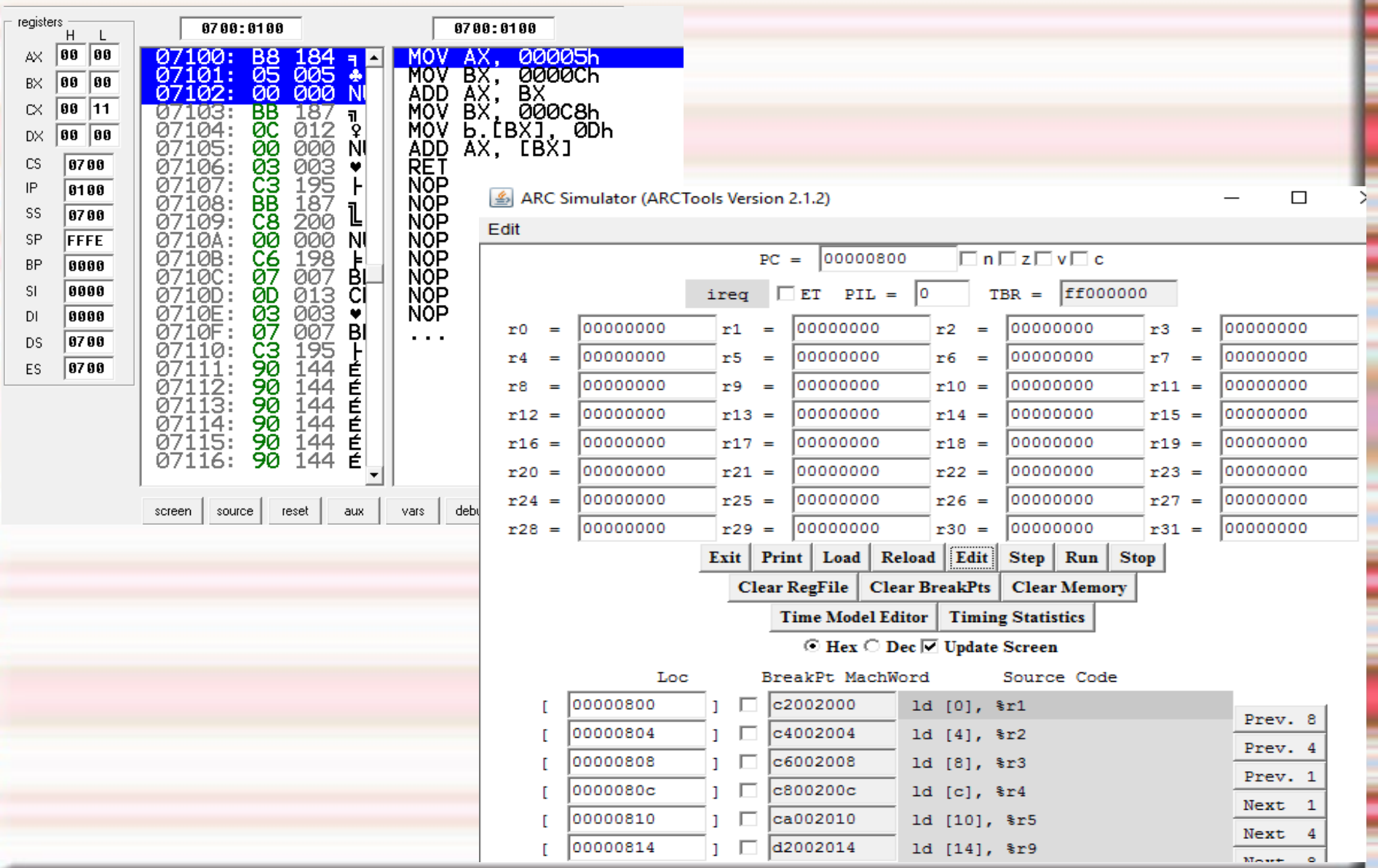

Typical access time
Typical capacity
1 nsec
2 nsec
10 nsec
10 msec
100 sec
Registers
Cache
Main memory
Magnetic disk
Magnetic tape
<1 KB
1 MB
64-512 MB
5-50 GB
20-100 GB

# CPU Components

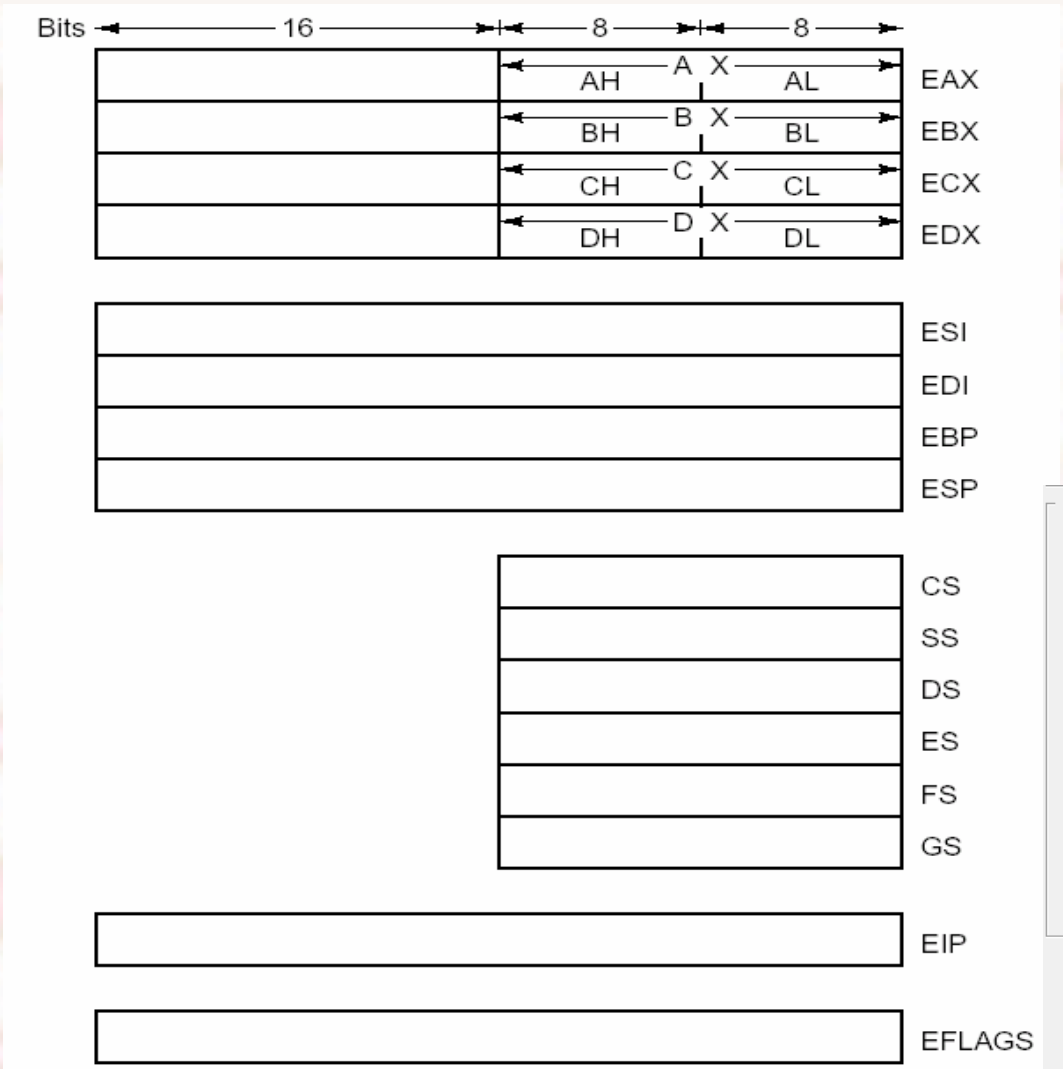
- **ALU** (Arithmetic Logical Unit)
  - performs arithmetic and logical operations.
- **IR – Instruction Register**
  - used to store the copy of the currently used instruction,
- **PC – Program Counter**
  - contains the address of the instruction currently used
- • **SP – Stack Pointer**
  - contains address of the ‘top’ of the certain memory region called ‘stack’,

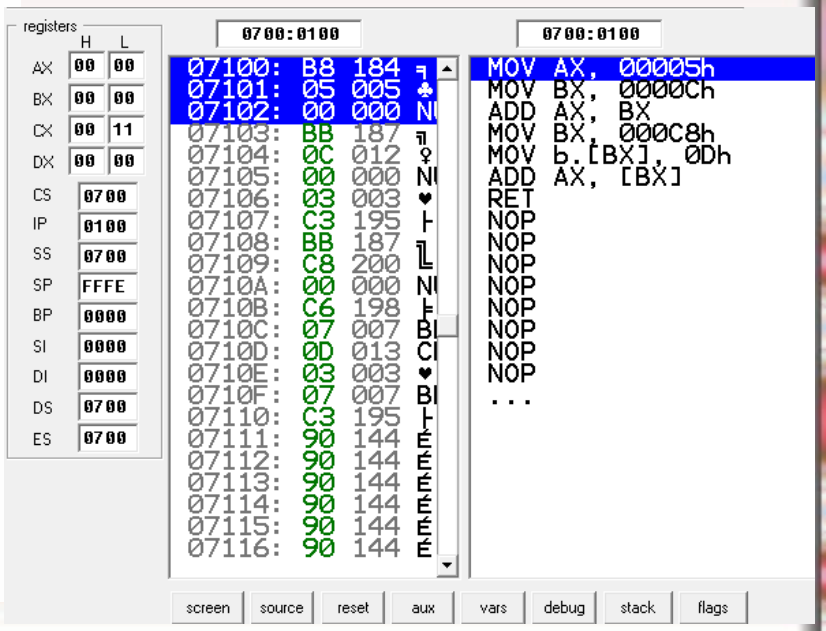
- **PS – Processor Status Register**
  - contains information including condition codes which allow CPU to perform ‘conditional jumps’ and other operations,
- **MAR – Memory Address Register**
  - contains the address of the memory location (RAM) to read or write to,
- • **MBR – Memory Buffer Register**
  - contains data to be written to or read from a memory location,
- • **DR – Data Registers**
  - storage places inside CPU where access is much faster than it is the case with using RAM.
- **XR – Index Register**
  - used with high-level programming languages that use arrays,

# Emulator Examples

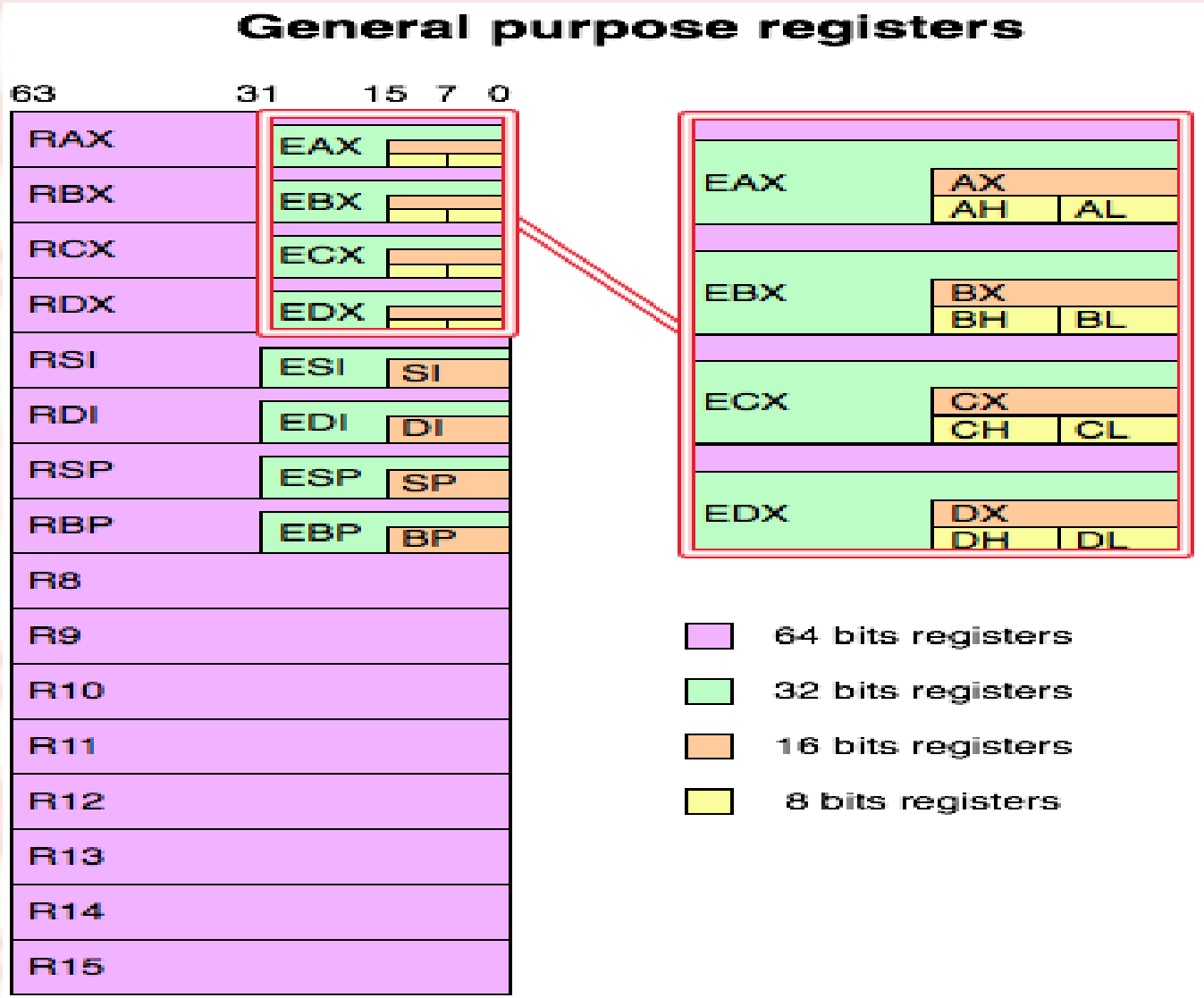


# Pentium 4 Primary Registers



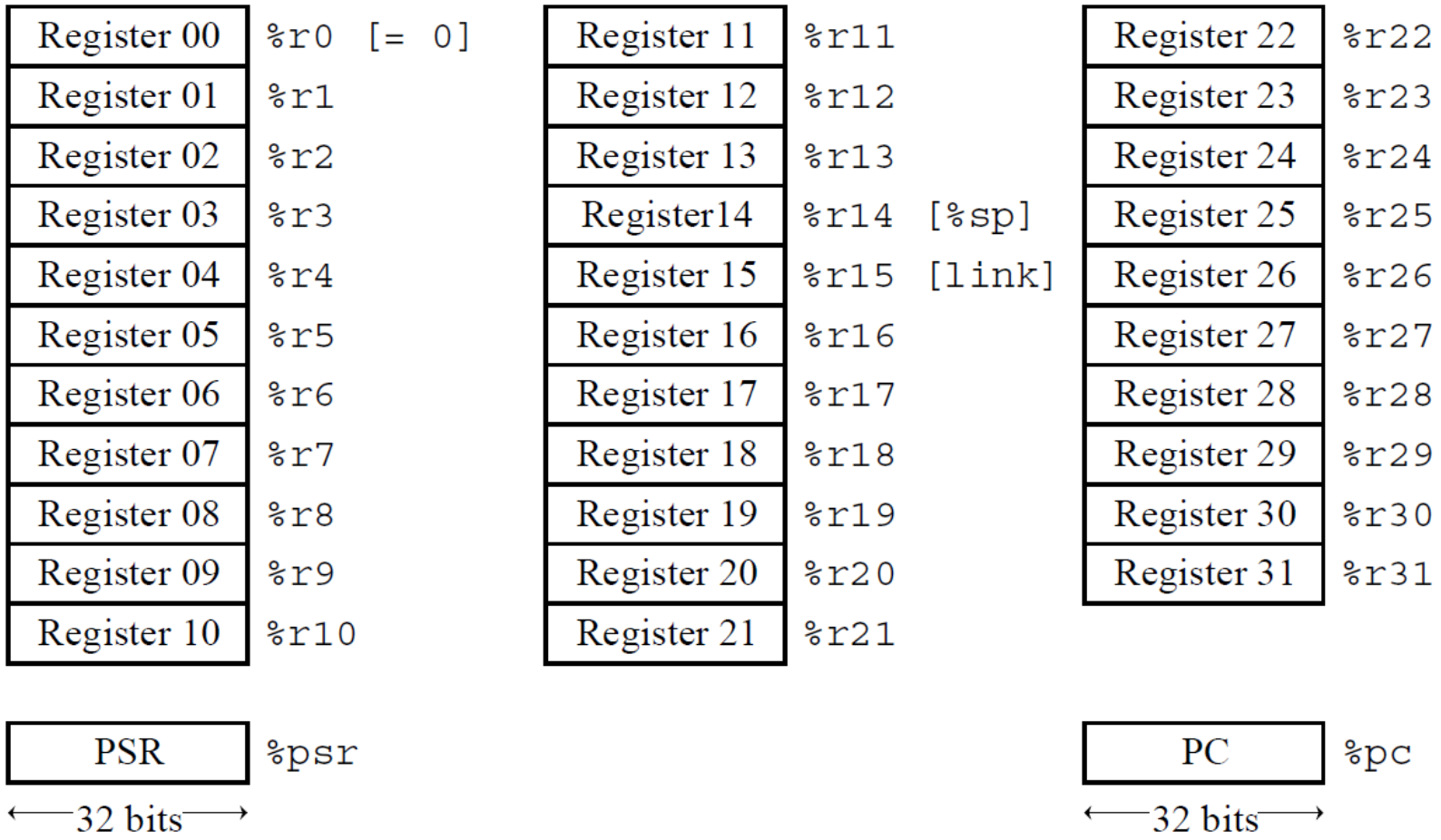



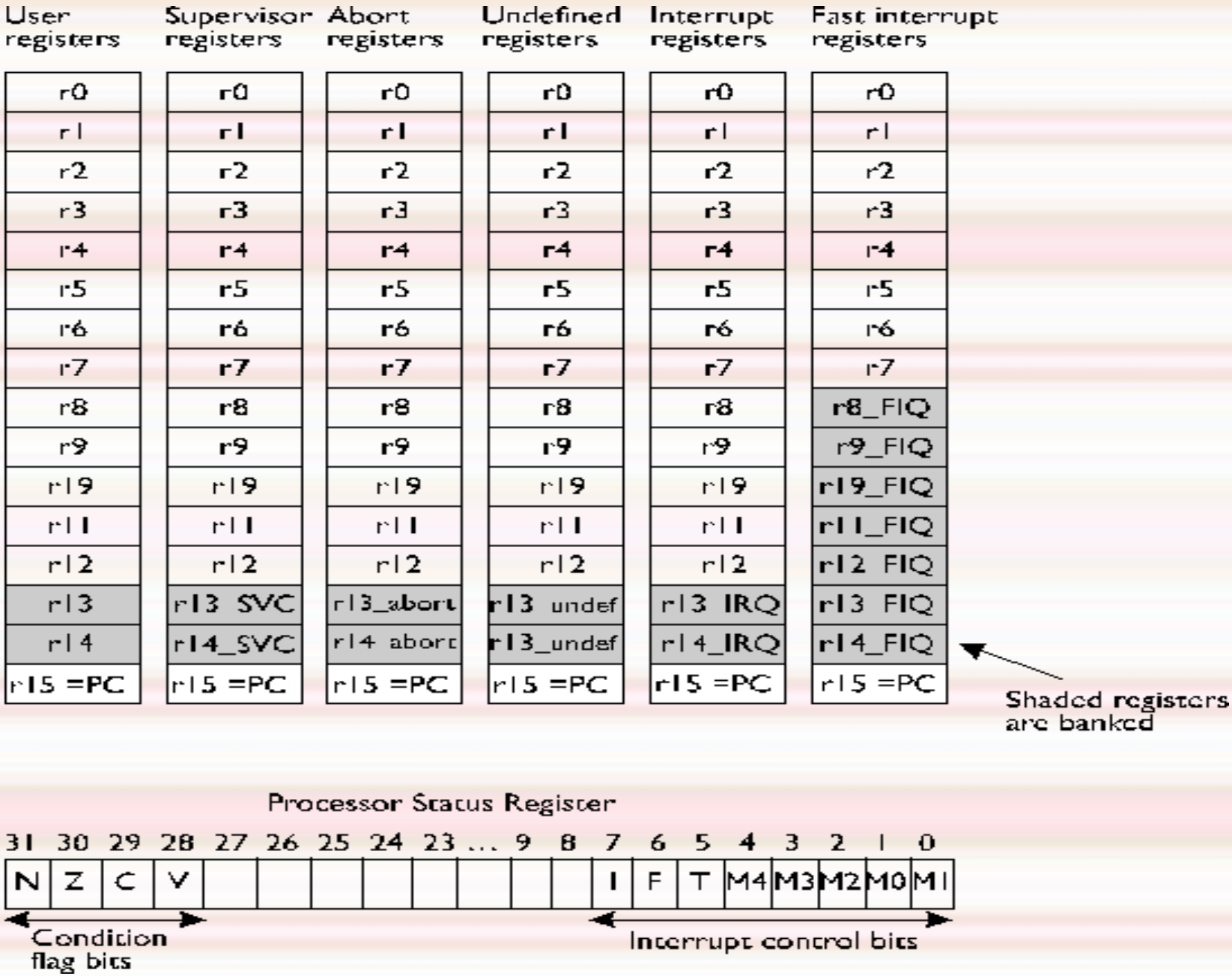
https://clementbera.files.wordpress.com/2014/01/gpreg.png



| Register | Bits | Purpose |
|---|---|---|
| EAX | 32 | Accumulator - Main arithmetic register. |
| EBX | 32 | Usually for base pointer (memory address). |
| ECX | 32 | Usually looping counter. |
| EDX | 32 | Used with EAX for holding 64-bit products and dividends. |
| ESI | 32 | Source index - pointer for the source string. |
| EDI | 32 | Destination index – pointer for the destination string. |
| EBP | 32 | Current stack frame base pointer. |
| ESP | 32 | Current stack pointer. |
| CS | 16 | 8088 compatible – Code segment register |
| SS | 16 | 8088 compatible – Stack segment register |
| DS | 16 | 8088 compatible – Data segment register |
| ES | 16 | 8088 compatible – Extra segment register |
| FS | 16 | 8088 compatible – Extra segment register |
| GS | 16 | 8088 compatible – Extra segment register |
| EIP | 32 | Program counter (Extended Instruction Pointer) |
| EFLAGS | 32 | PSW (Program Status Word) |

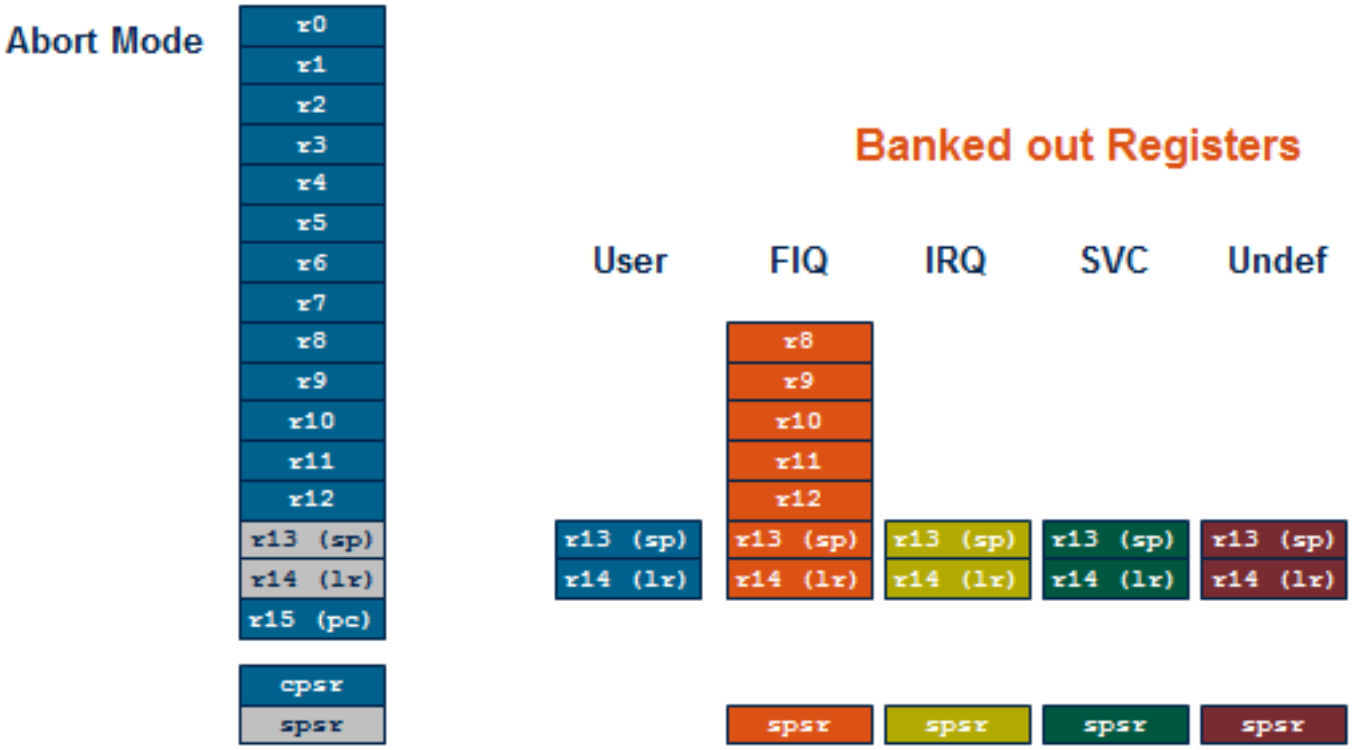
# ARC User-Visible Registers




User registers
Supervisor registers
Abort registers
Undefined registers
Interrupt registers
Fast interrupt registers
r0
r1
r2
r3
r4
r5
r6
r7
r8
r9
r10
r11
r12
r13
r14
r15 =PC
r13_SVC
r14_SVC
r13_abort
r14_abort
r13_undef
r14_undef
r13_IRQ
r14_IRQ
r8_FIQ
r9_FIQ
r10_FIQ
r11_FIQ
r12_FIQ
r13_FIQ
r14_FIQ
Shaded registers are banked
Processor Status Register
31 30 29 28 27 26 25 24 23 ... 9 8 7 6 5 4 3 2 1 0
N Z C V
I F T M4 M3 M2 M0 M1
Condition flag bits
Interrupt control bits

# ARM The ARM Register Set

## Current Visible Registers



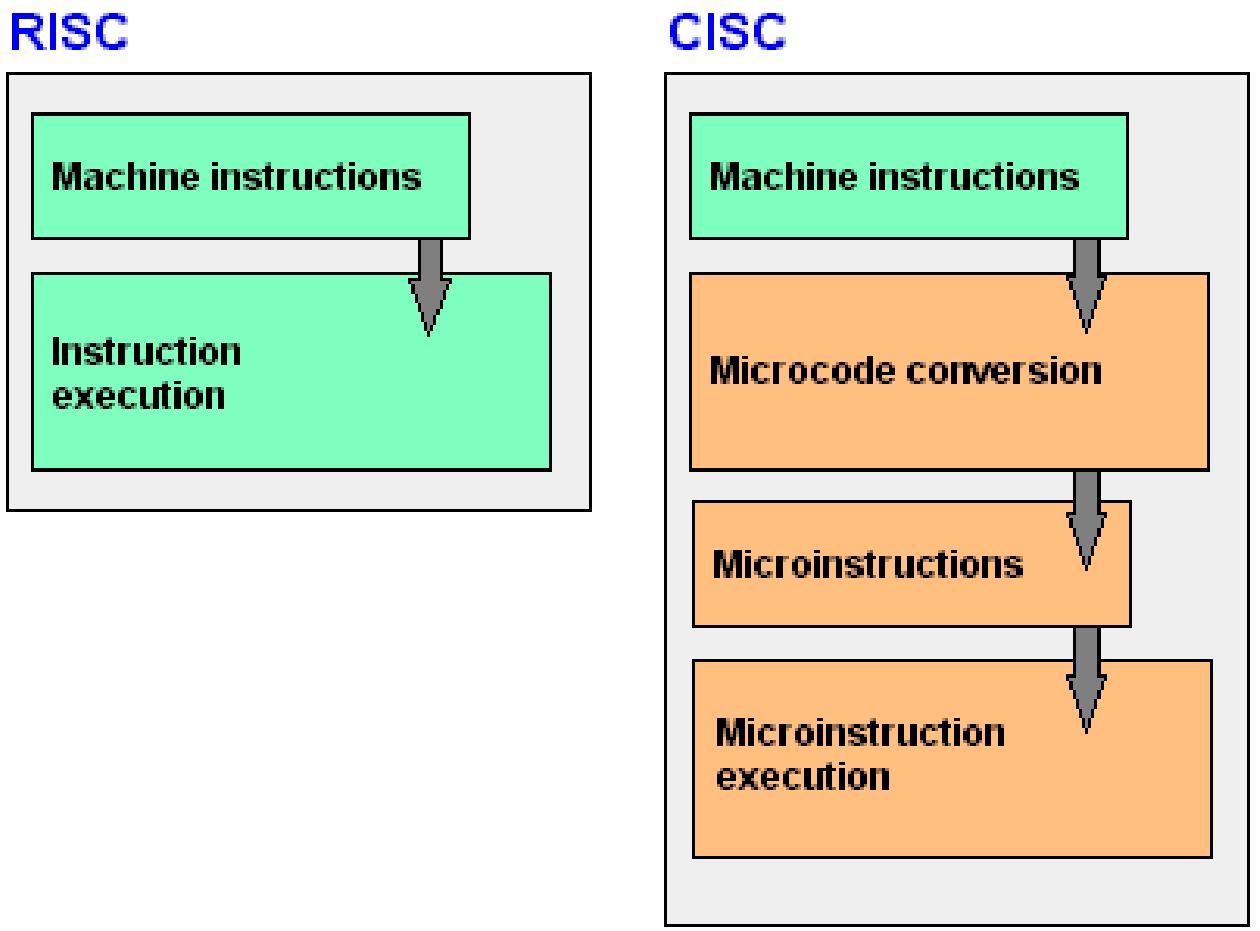
# RISC and CISC Architectures

- **CISC** (Complex Instruction Set Computer)
  - Large number of interpreted instructions, usually 200 – 300 (Intel, IBM mainframes).
- Each CISC instruction is broken down to a number of *microinstructions* requiring one or more cycles to complete.
- CISC architecture (usually) preserves backwards compatibility within the same family of CPU-s.
  - No longer true…

# RISC

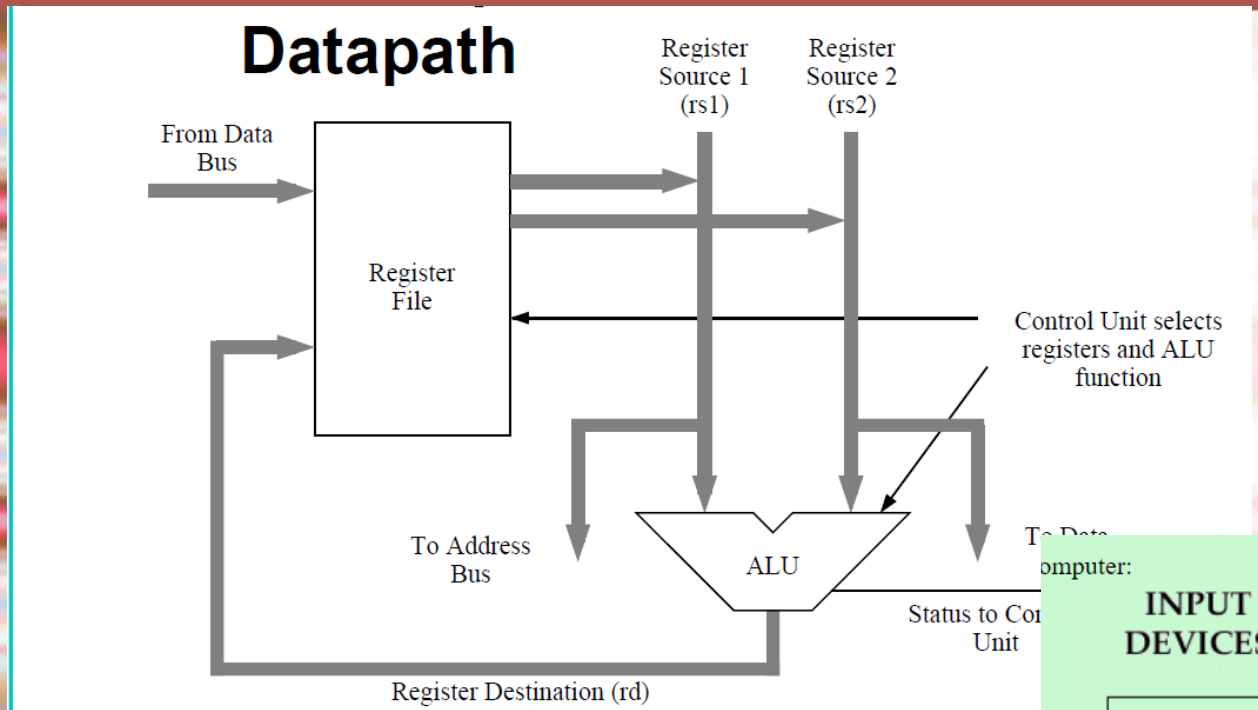
- **RISC** (Reduced Instruction Set Computer), small number (around 50) of simple instructions that execute in one cycle each.
  - In 1980 Patterson and Sequin designed CPU chips RISC I, and RISC II.
  - In 1981 John Hennessy designed MIPS chip that evolved into SPARC.
- RISC instructions are not interpreted.

From Computer Desktop Encyclopedia
© 1998 The Computer Language Co. Inc.



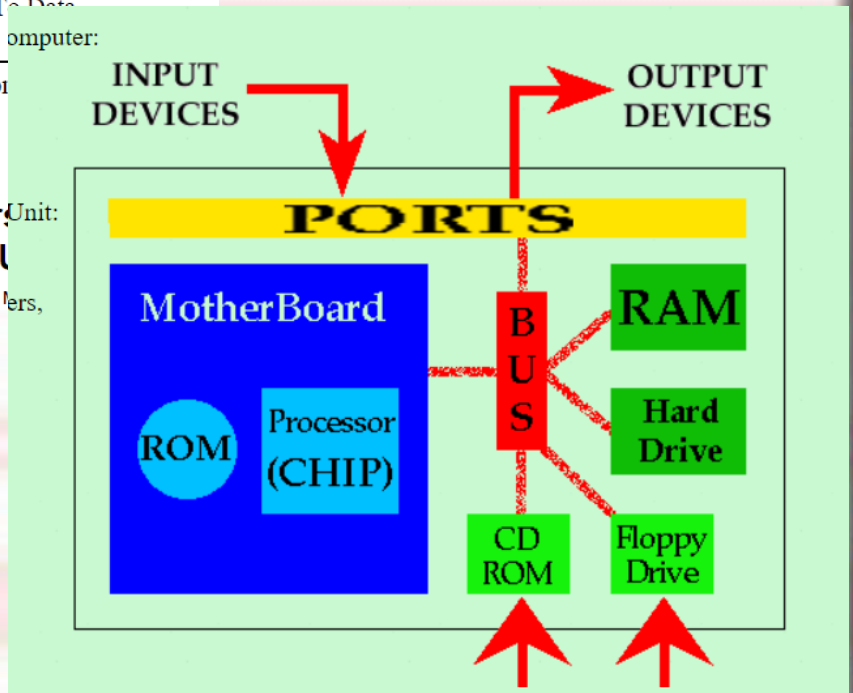
# Design Principles for Modern Computers

- **1. All instructions are directly executed by hardware.**
  - This provides high speed for most of the instructions.
  - More complex CISC instructions may be broken into separate parts similar to microinstructions.
- **2. Instructions need to be easy to decode.**
- **3. Only LOAD and STORE instructions should reference memory.**
- **4. Provide enough CPU memory registers.**




- **The ARC datapath is made up of a collection of register as the *register file* and the *arithmetic and logic unit* (ALU**

*Principles of Computer Architecture* by M. Murdocca and V. Heuring © 1999 M.




https://homepage.cs.uri.edu/faculty/wolfe/book/Readings/Reading04.htm

# The ARC ISA

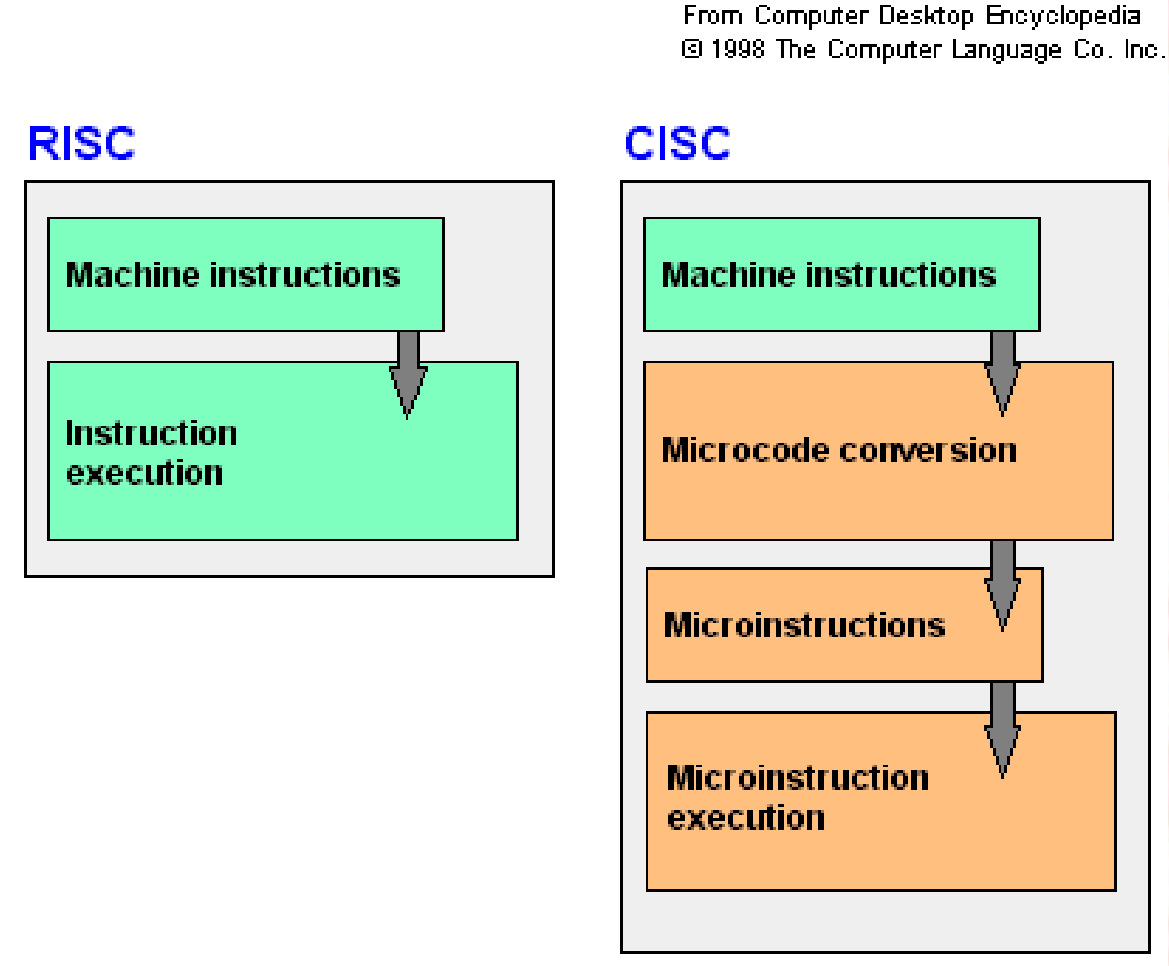
- **The ARC ISA is a subset of the SPARC ISA.**

| | Mnemonic | Meaning |
|---|---|---|
| Memory | **ld** | Load a register from memory |
| | **st** | Store a register into memory |
| Logic | **sethi** | Load the 22 most significant bits of a register |
| | **andcc** | Bitwise logical AND |
| | **orcc** | Bitwise logical OR |
| | **orncc** | Bitwise logical NOR |
| | **srl** | Shift right (logical) |
| Arithmetic | **addcc** | Add |
| Control | **call** | Call subroutine |
| | **jmpl** | Jump and link (return from subroutine call) |
| | **be** | Branch if equal |
| | **bneg** | Branch if negative |
| | **bcs** | Branch on carry |
| | **bvs** | Branch on overflow |
| | **ba** | Branch always |

# Complete 8086 Instruction Set

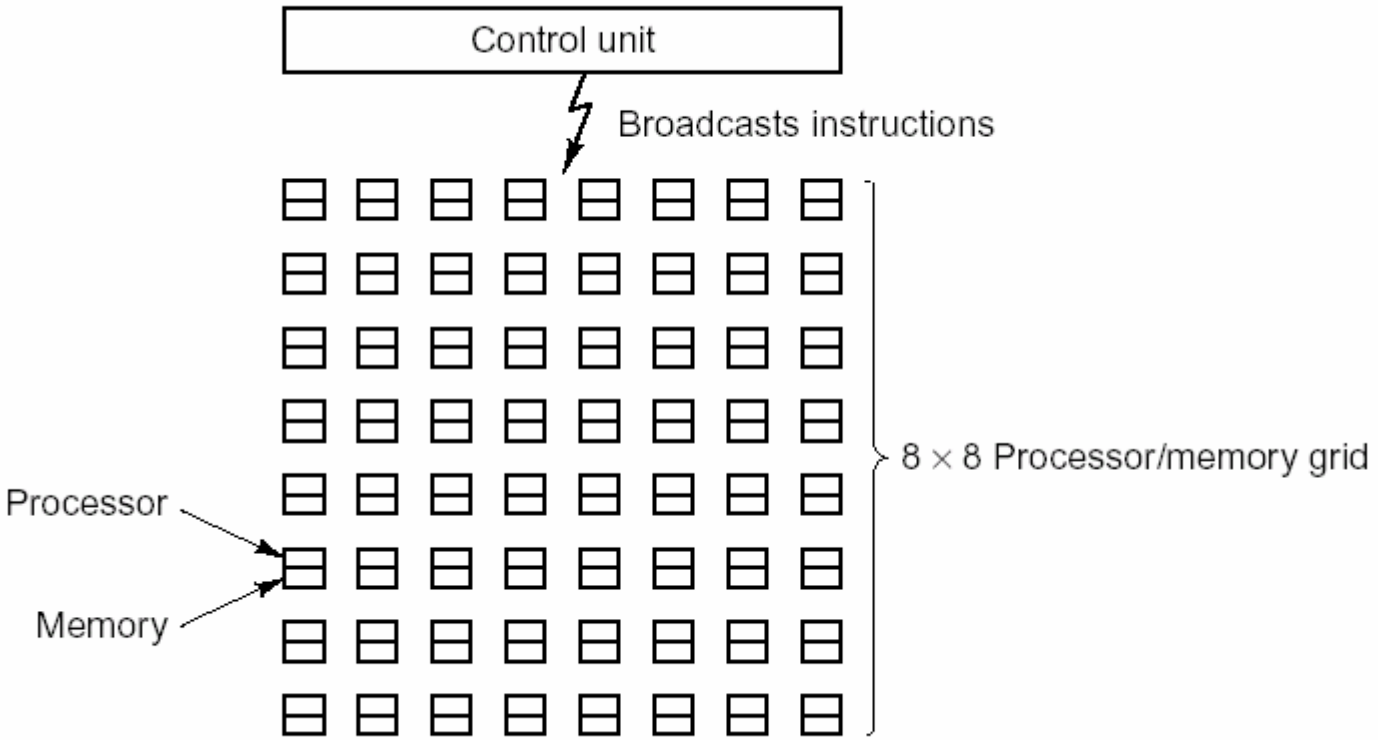
| | | | | | | | |
|---|---|---|---|---|---|---|---|
| | CMPSB | | | | MOV | | |
| AAA | CMPSW | JAE | JNBE | JPO | MOVSB | RCR | SCASB |
| AAD | CWD | JB | JNC | JS | MOVSW | REP | SCASW |
| AAM | DAA | JBE | JNE | JZ | MUL | REPE | SHL |
| AAS | DAS | JC | JNG | LAHF | NEG | REPNE | SHR |
| ADC | DEC | JCXZ | JNGE | LDS | NOP | REPNZ | STC |
| ADD | DIV | JE | JNL | LEA | NOT | REPZ | STD |
| AND | HLT | JG | JNLE | LES | OR | RET | STI |
| CALL | IDIV | JGE | JNO | LODSB | OUT | RETF | STOSB |
| CBW | IMUL | JL | JNP | LODSW | POP | ROL | STOSW |
| CLC | IN | JLE | JNS | LOOP | POPA | ROR | SUB |
| CLD | INC | JMP | JNZ | LOOPE | POPF | SAHF | TEST |
| CLI | INT | JNA | JO | LOOPNE | PUSH | SAL | XCHG |
| CMC | INTO | JNAE | JP | LOOPNZ | PUSHA | SAR | XLATB |
| CMP | IRET | JNB | JPE | LOOPZ | PUSHF | SBB | XOR |
| | JA | | | | RCL | | |

# The RISC chip is faster than its CISC counterpart and is designed and built more economically



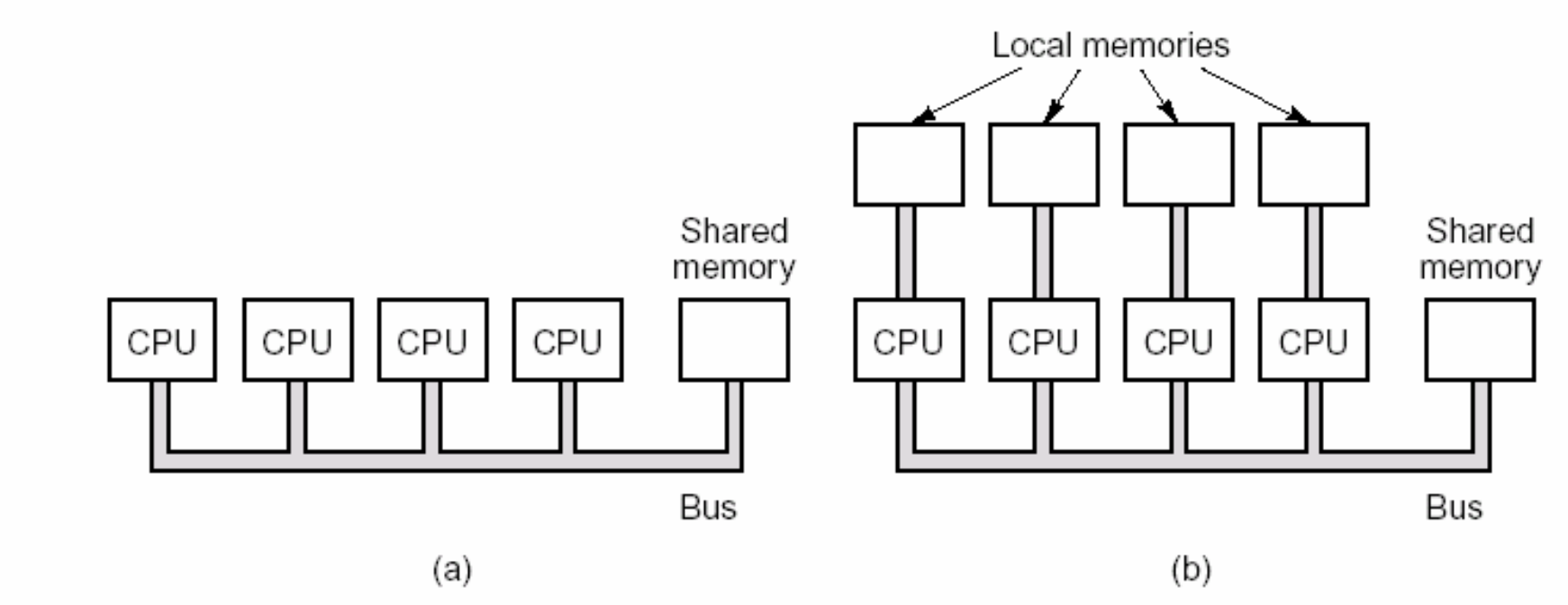
# Processor-Level Parallelism

- Array of Computers




An array processor of the ILLIAC IV type

# Multiprocessors




(a) A single-bus multiprocessor; (b) A multiprocessor with local memories

# discussions

- Provide register names for Program counter in intel/ARC/ARM processors.
- Provide register names for storing Processor states in intel/ARC/ARM processors.
- Provide register names for  storing the top of the Stack in intel/ARC/ARM processors.
- Describe the Design Principles for Modern Computers

# Discussion (2)

- Describe the general differences between CISC and RISC computers…
- Provide the names of computers that implemented the CISC architecture…
- Provide the names of the computer implemented the RISC architecture…

# Primary Memory

- The basic unit of memory is the binary digit
  - called a **bit**
- Memory consists of bytes called **cells**
  - can be used to store predefined number of bits.
  - Cell is a smallest addressable unit of memory with a unique numerical **address**.
  - Eight-bit cell is called **byte**
    - bytes are grouped into **words**
  - Most computers allow more than one byte to be loaded or stored at one time.

- A load or store operation is performed with data equal in size to the system's bit width.
  - The address sent to the memory system is the lowest-addressed byte of data to be manipulated.
    - For example, a 32-bit system loads (stores) 32 bits (4 bytes) of data with each operation's address.
    - A load from location 4804 would return bit-content of four consecutive bytes in memory: 4804, 4805, 4806 and 4807.
- Some computers require loads and stores
  - to be 'aligned'
    - i.e. address of a memory reference must be multiple of the size of manipulated data.

# Common Sizes for Data Types

| | |
|---|---|
| Bit | 0 |
| Nibble | 0110 |
| Byte | 10110000 |
| 16-bit word (halfword) | 11001001 01000110 |
| 32-bit word | 10110100 00110101 10011001 01011000 |
| 64-bit word (double) | 01011000 01010101 10110000 11110011<br>11001110 11101110 01111000 00110101 |
| 128-bit word (quad) | 01011000 01010101 10110000 11110011<br>11001110 11101110 01111000 00110101<br>00001011 10100110 11110010 11100110<br>10100100 01000100 10100101 01010001 |

| Type | Storage size | Value range |
|---|---|---|
| **char** | 1 byte | -128 to 127 or 0 to 255 |
| **unsigned char** | 1 byte | 0 to 255 |
| **signed char** | 1 byte | -128 to 127 |
| **int** | 2 or 4 bytes | -32,768 to 32,767 or -2,147,483,648 to 2,147,483,647 |
| **unsigned int** | 2 or 4 bytes | 0 to 65,535 or 0 to 4,294,967,295 |
| **short** | 2 bytes | -32,768 to 32,767 |
| **unsigned short** | 2 bytes | 0 to 65,535 |
| **long** | 8 bytes | -9,223,372,036,854,775,808 to 9,223,372,036,854,775,80 |
| **unsigned long** | 8 bytes | 0 to 18,446,745,073,709,551,616 |

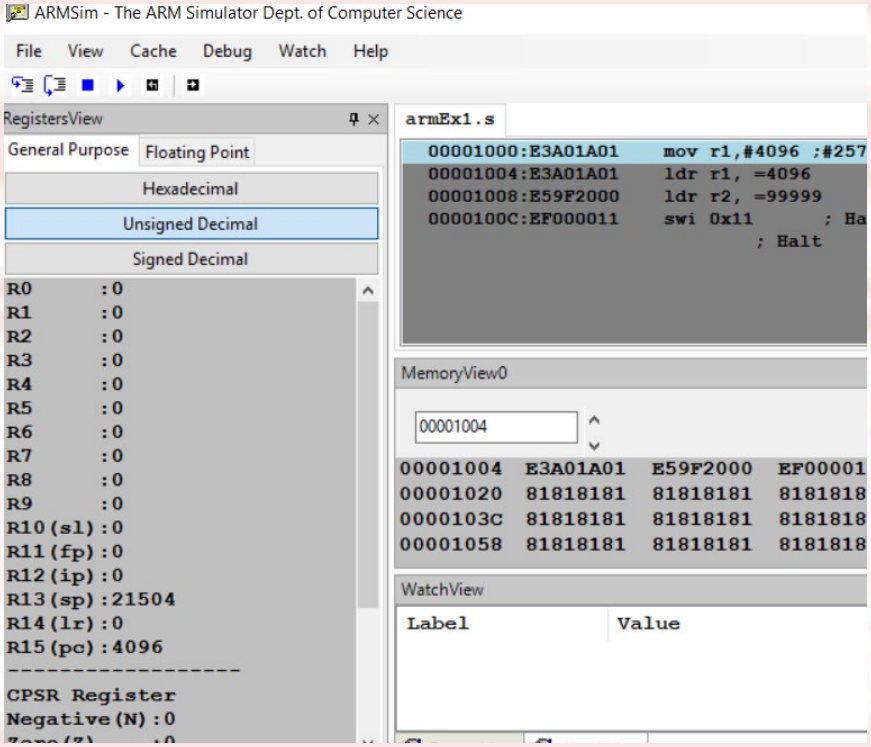
# Ariane 5

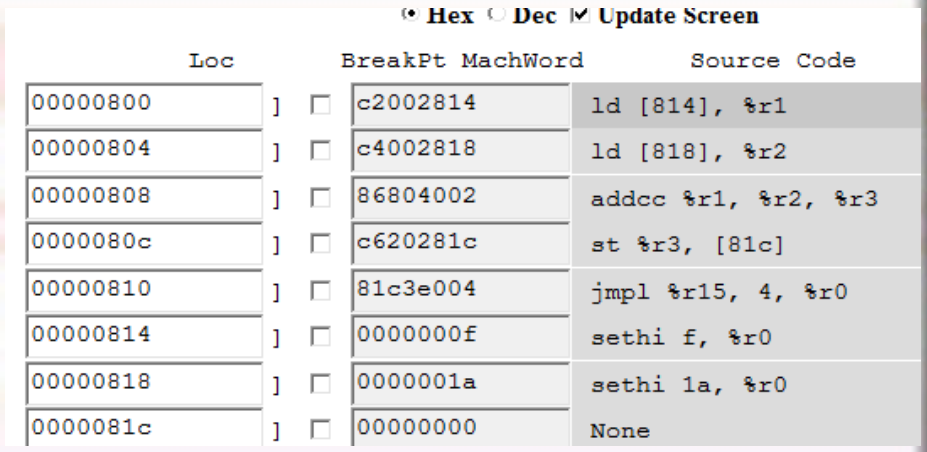
- https://www.youtube.com/watch?v=gp_D8r-2hwk

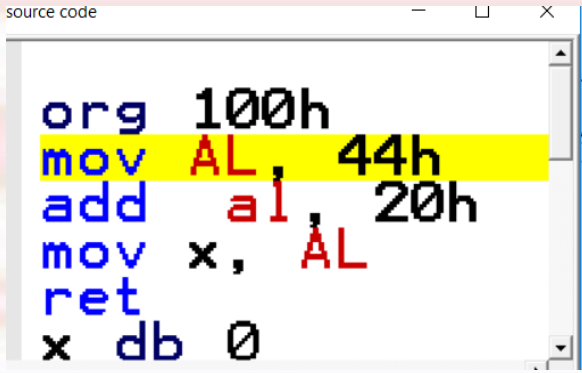
# Hex decimal

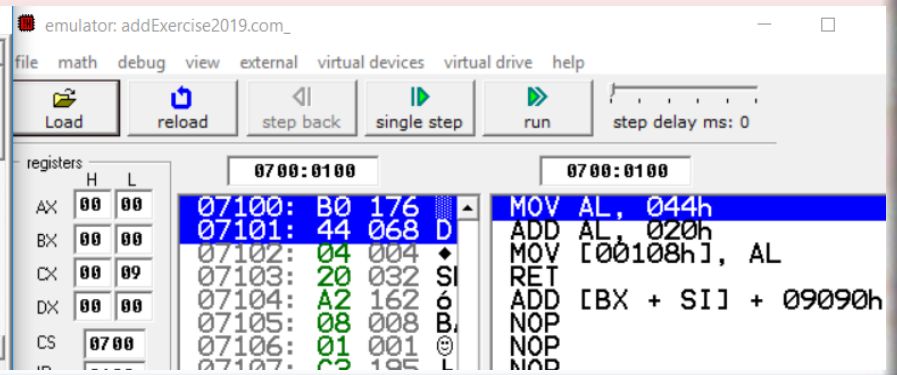
| Decimal | Binary | Hexadecimal |
|---|---|---|
| 0 | 0000 | 0 |
| 1 | 0001 | 1 |
| 2 | 0010 | 2 |
| 3 | 0011 | 3 |
| 4 | 0100 | 4 |
| 5 | 0101 | 5 |
| 6 | 0110 | 6 |
| 7 | 0111 | 7 |
| 8 | 1000 | 8 |
| 9 | 1001 | 9 |
| 10 | 1010 | A |
| 11 | 1011 | B |
| 12 | 1100 | C |
| 13 | 1101 | D |
| 14 | 1110 | E |
| 15 | 1111 | F |

# Hex decimal


ARMSim - The ARM Simulator Dept. of Computer Science
File View Cache Debug Watch Help
RegistersView
General Purpose Floating Point
Hexadecimal
Unsigned Decimal
Signed Decimal
R0 :0
R1 :0
R2 :0
R3 :0
R4 :0
R5 :0
R6 :0
R7 :0
R8 :0
R9 :0
R10 (sl) :0
R11 (fp) :0
R12 (ip) :0
R13 (sp) :21504
R14 (lr) :0
R15 (pc) :4096
CPSR Register
Negative (N) :0
armEx1.s
00001000:E3A01A01 mov r1,#4096 ;#257
00001004:E3A01A01 ldr r1, =4096
00001008:E59F2000 ldr r2, =99999
0000100C:EF000011 swi 0x11 ; Ha
; Halt
MemoryView0
00001004
00001004 E3A01A01 E59F2000 EF00001
00001020 81818181 81818181 8181818
0000103C 81818181 81818181 8181818
00001058 81818181 81818181 8181818
WatchView
Label Value

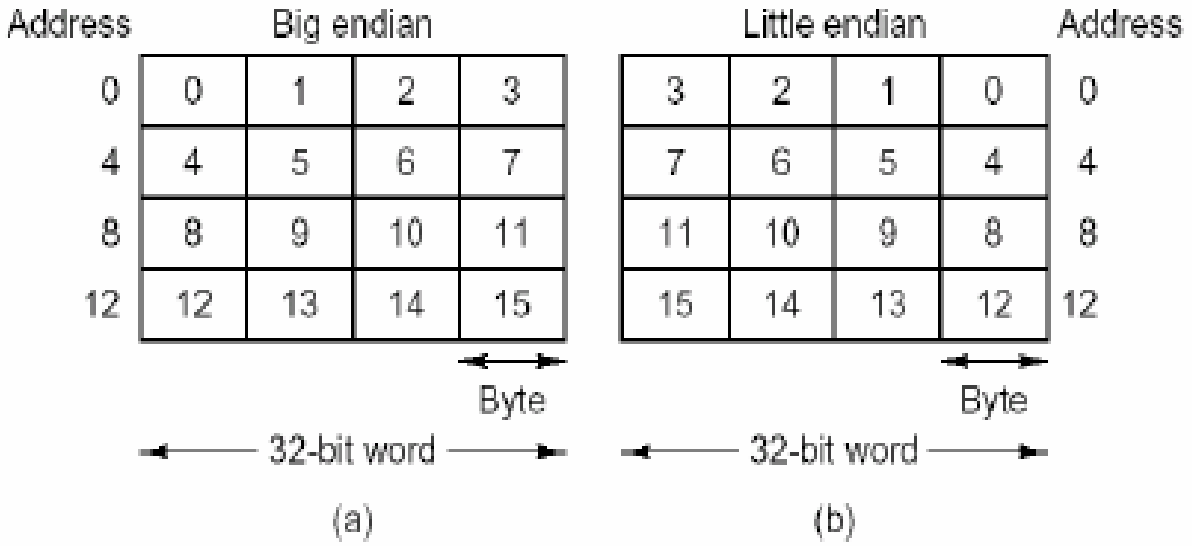

Hex Dec Update Screen
Loc BreakPt MachWord Source Code
00000800 c2002814 ld [814], %r1
00000804 c4002818 ld [818], %r2
00000808 86804002 addcc %r1, %r2, %r3
0000080c c620281c st %r3, [81c]
00000810 81c3e004 jmpl %r15, 4, %r0
00000814 0000000f sethi f, %r0
00000818 0000001a sethi 1a, %r0
0000081c 00000000 None


source code
org 100h
mov AL, 44h
add al, 20h
mov x, AL
ret
x db 0

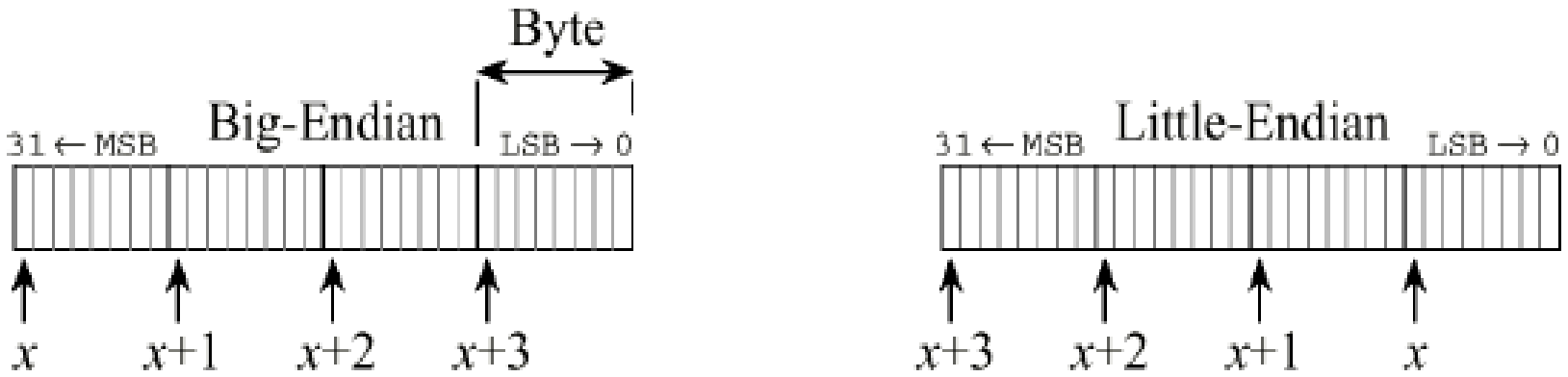

emulator: addExercise2019.com_
file math debug view external virtual devices virtual drive help
Load reload step back single step run step delay ms: 0
registers
AX BX CX DX CS
0700:0100
07100: B0 176
07101: 44 068 D
07102: 04 004
07103: 20 032 SI
07104: A2 162 ó
07105: 08 008 B
07106: 01 001
MOV AL, 044h
ADD AL, 020h
MOV [00108h], AL
RET
ADD [BX + SI] + 09090h
NOP
NOP

# Big-Endian and Little-Endian Formats

- In a byte-addressable machine
  - the smallest amount of data that can be referenced in memory is **byte**.
- Multi-byte words are stored as a sequence of bytes
  - in which the address of the multi-byte word is the same as the byte of the word that has the lowest address.
- **Byte Ordering**
  - From left to right – **big endian** (SPARC, Motorola, internet data transfer).
  - From right to left – **little endian** (Intel)..




(a) Big endian memory; (b) Little endian memory




Word address is $x$ for both big-endian and little-endian formats.

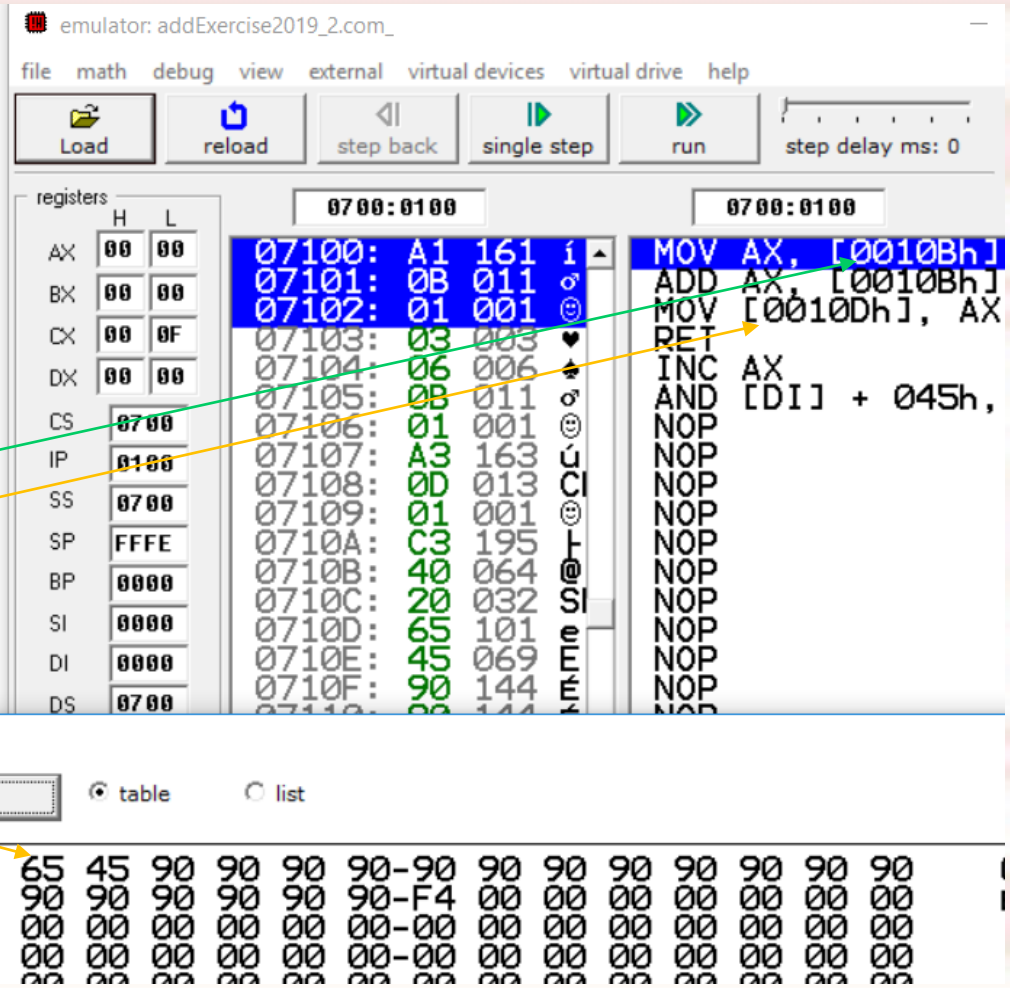
- *Big endian*
  - The most significant byte of a word is written into the lowest-addressed byte, and the other bytes are written in a decreasing order of significance.
- *Little endian*
  - The least significant byte of a word is written into the lowest-addressed byte, and the other bytes are written in an increasing order of significance.

Word = 0x123abcde
Address = 0x135a

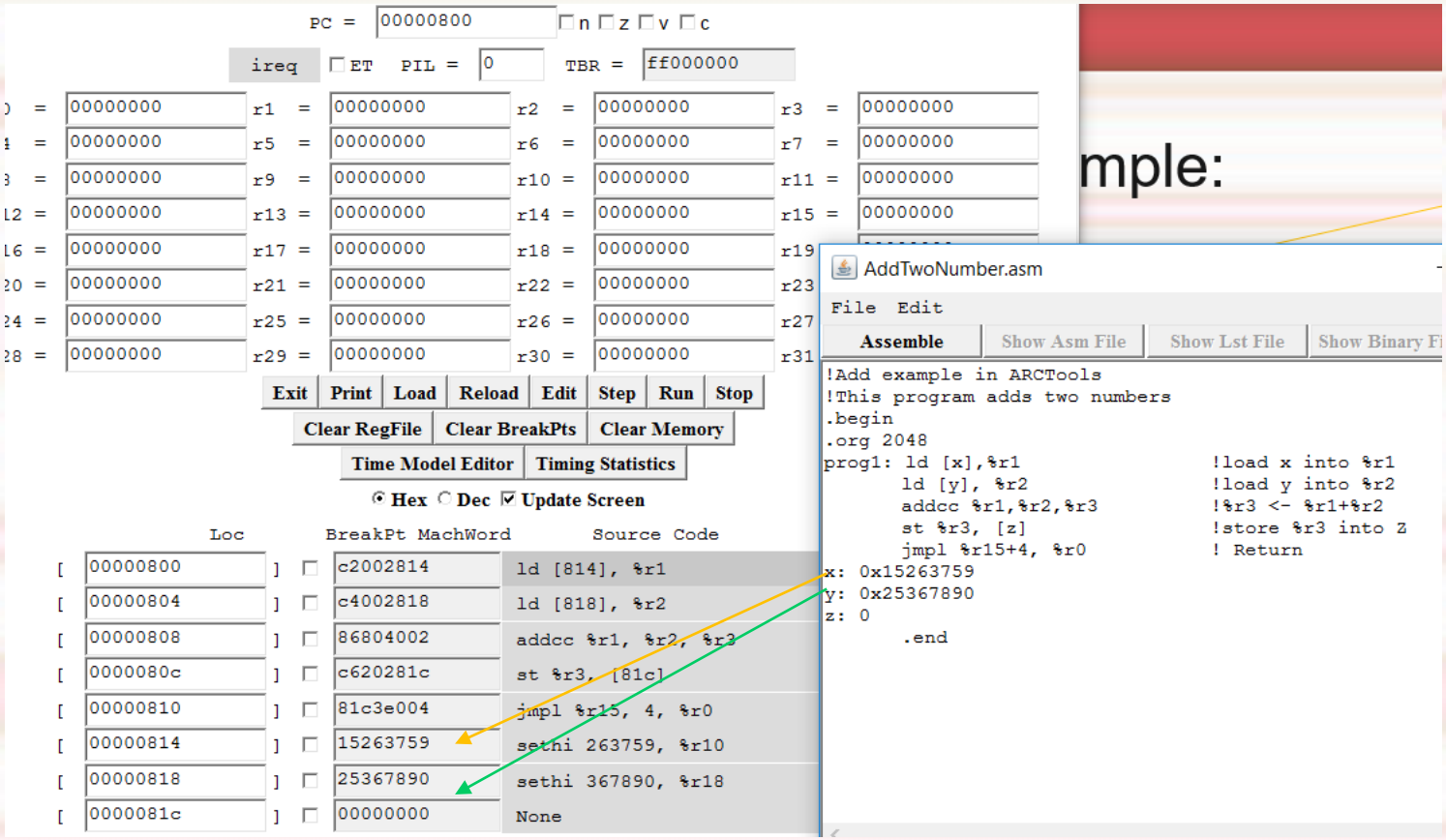
| | 0x135a | 0x135b | 0x135c | 0x135d |
|---|---|---|---|---|
| Big endian | 12 | 3a | bc | de |
| Little endian | de | bc | 3a | 12 |

# Little-Endian example:

```
org  100h

mov  ax, x
add   ax, x
mov  y,  Ax

ret


x  dw  2040h
y  dw  4565h
```



# Big-Endian example:



Ex:

- The data word is stored in memory as below:
- Memory Addr.: value in this memory cell:

| | |
|---|---|
| 60012 | 18 |
| 60013 | 2C |
| 60014 | 7F |
| 60015 | 39 |

- What is the value represented by “Big-Endian” format and what is the value represented by “Little-Endian” format?

# ex2

- An integer data in the cpu is:

  A1 B2 C3 D4

- We want to store the data to memory
- how is this data stored when using

  Little-Endian?

- how is this data stored when using

  Big-Endian?

# Cache Memory

- **Locality principle**
  - memory references in any short time interval tend to use a small fraction of the total memory.
  - Mean access time = c + (1 – h)m
  - c is cache access time, m is main memory access time, and h is hit ratio, i.e. fraction of all references that can be satisfied by what is in the cache memory.

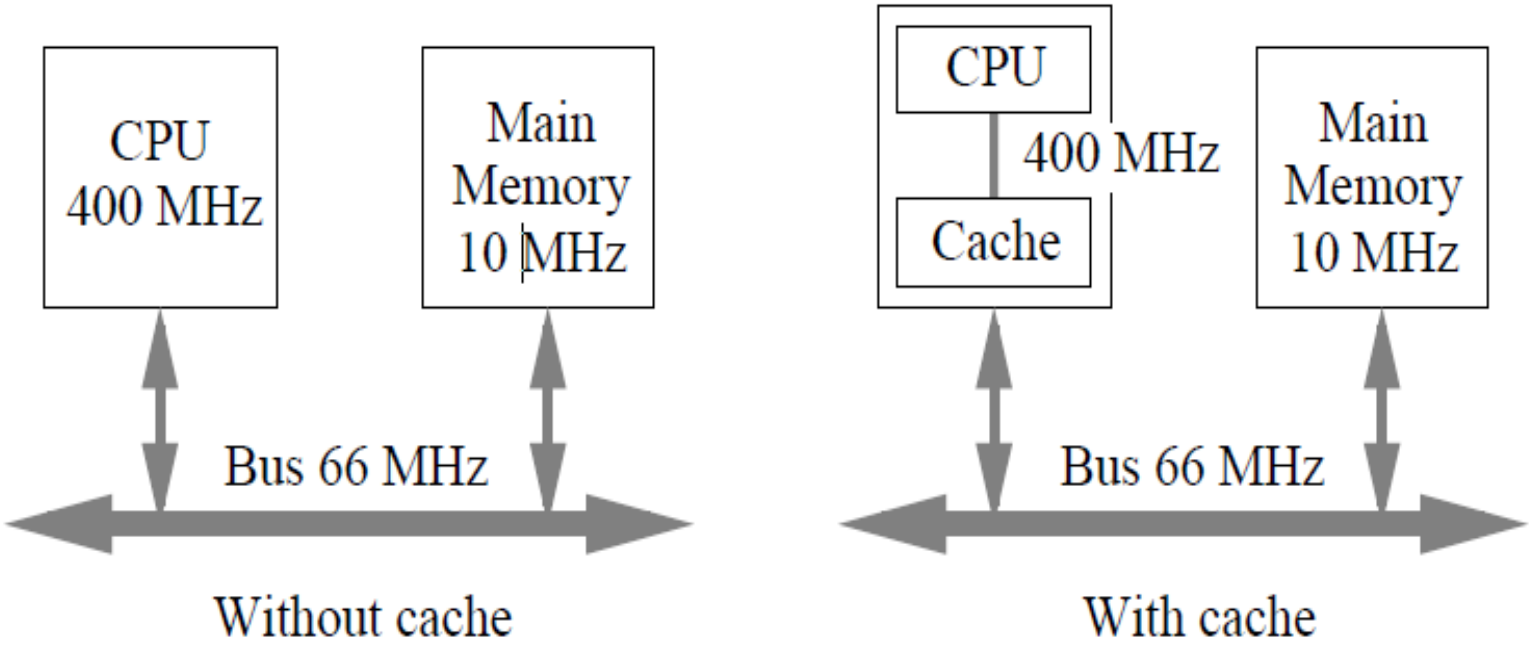
# principle of locality

- **Locality of reference**
  - In computer science, **locality of reference**
    - the phenomenon of the same value or related storage locations being frequently accessed.
  - There are two basic types of reference locality
    - Temporal locality
      - the reuse of specific data and/or resources within relatively small time durations
    - Spatial locality
      - the use of data elements within relatively close storage locations.

# Principle of Locality

- Programs tend to reuse data and instructions they have used recently
  - A rule of thumb is that **a program spends 90% of its execution time in only 10% of the code**
  - ***Temporal locality***
    - recently accessed items are likely to be accessed in the near future
  - ***Spatial locality***
    - items whose addresses are near one another tend to be referenced close together in time

# Placement of Cache in a Computer System

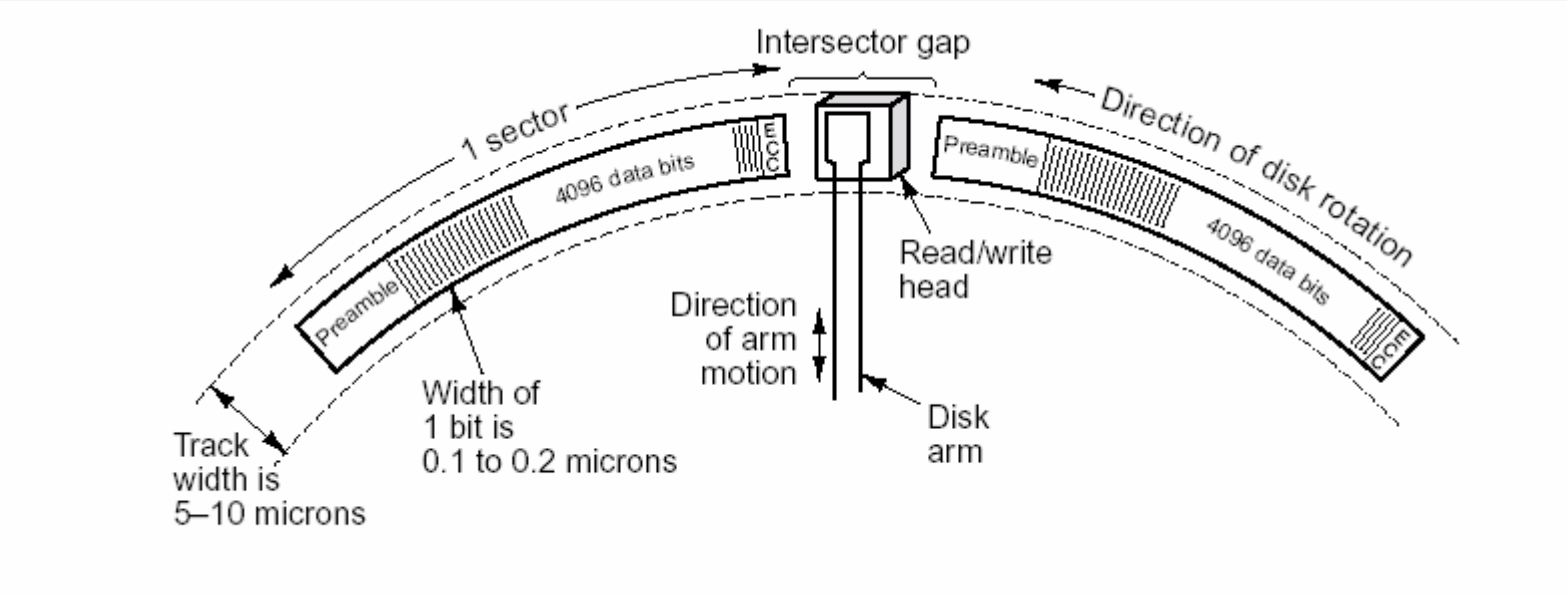


# MAT Exercise

- m = 0.98ms c = 0.1ms h = 0.89 what will be the MAT?

# Secondary Memory

## Magnetic Disks




Two sectors of the disk track

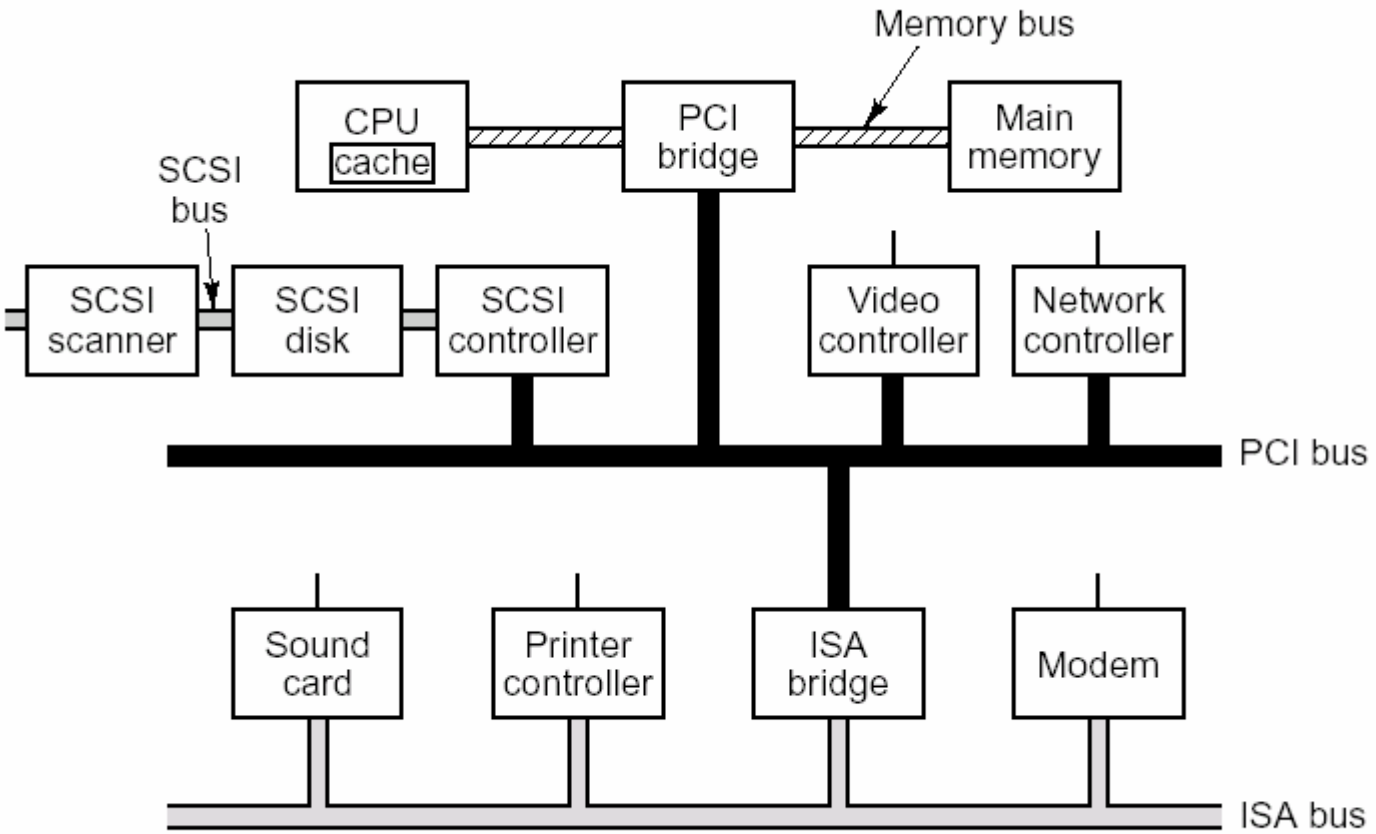
- **Disk performance depends on:**
- 1. Average seek time
  - between random tracks – radial head positioning
- 2. Rotational latency
  - time for the desired sector to pass under the head
- 3. Transfer time
  - the linear density and rotational speed
  - Each disk drive has its own **disk controller**

# Different types of disks:

- • Floppy disks,
- • IDE (Integrated Drive Electronics) disk drives,
- • EIDE (Extended IDE) disk drives,
- • SCSI (Small Computer System Interface) disk drives,
- • RAID (Redundant Array of Inexpensive Disks),
- • CD-ROMs
- • CD-Rs
- • CD-RWs
- • DVDs

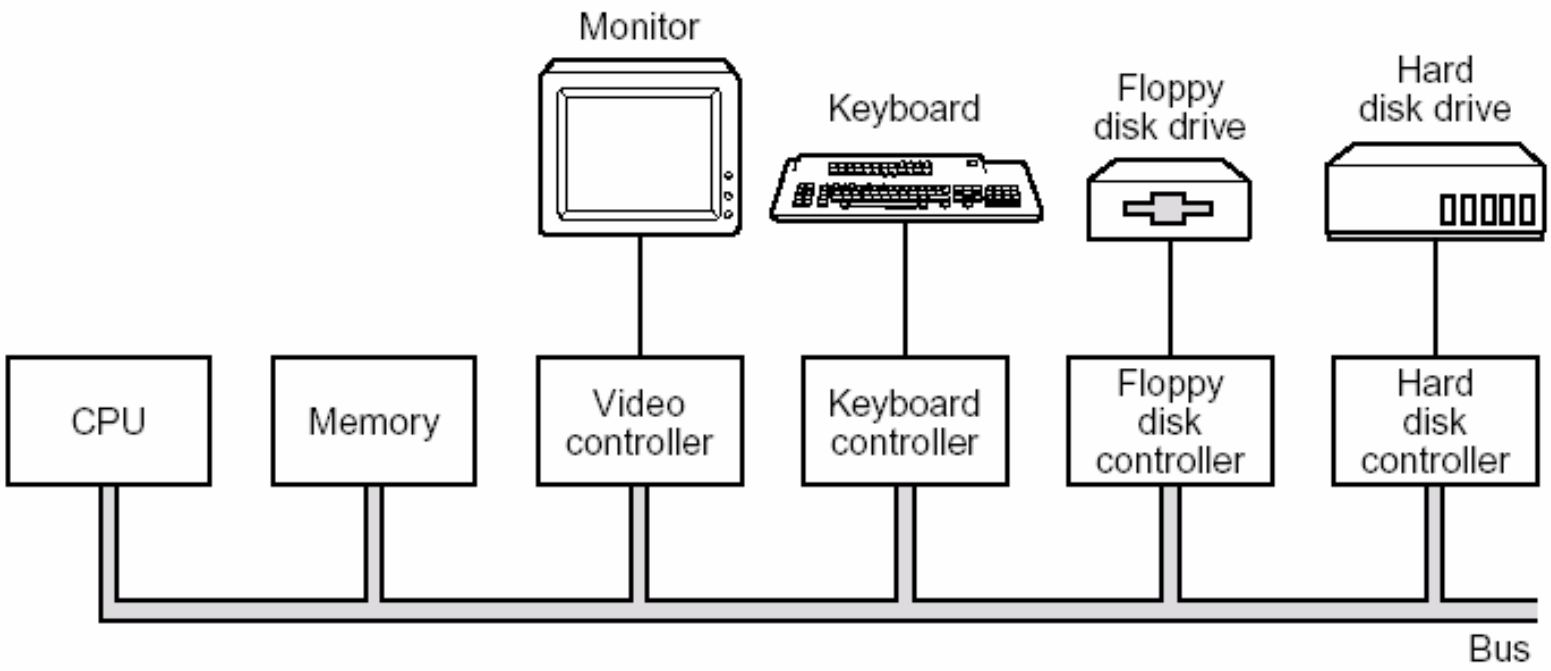
# Input/Output

- Motherboard
- Controllers/Drivers
- Bus arbiter
- Busses:
  - ISA (Industry Standard Architecture)
  - EISA (Extended ISA)
  - PCI (Peripheral Component Interconnect)
- Most PCs come with multiple busses: *ISA for slower, and PCI for faster devices*
- PCI bus also has a bridge to the ISA bus.

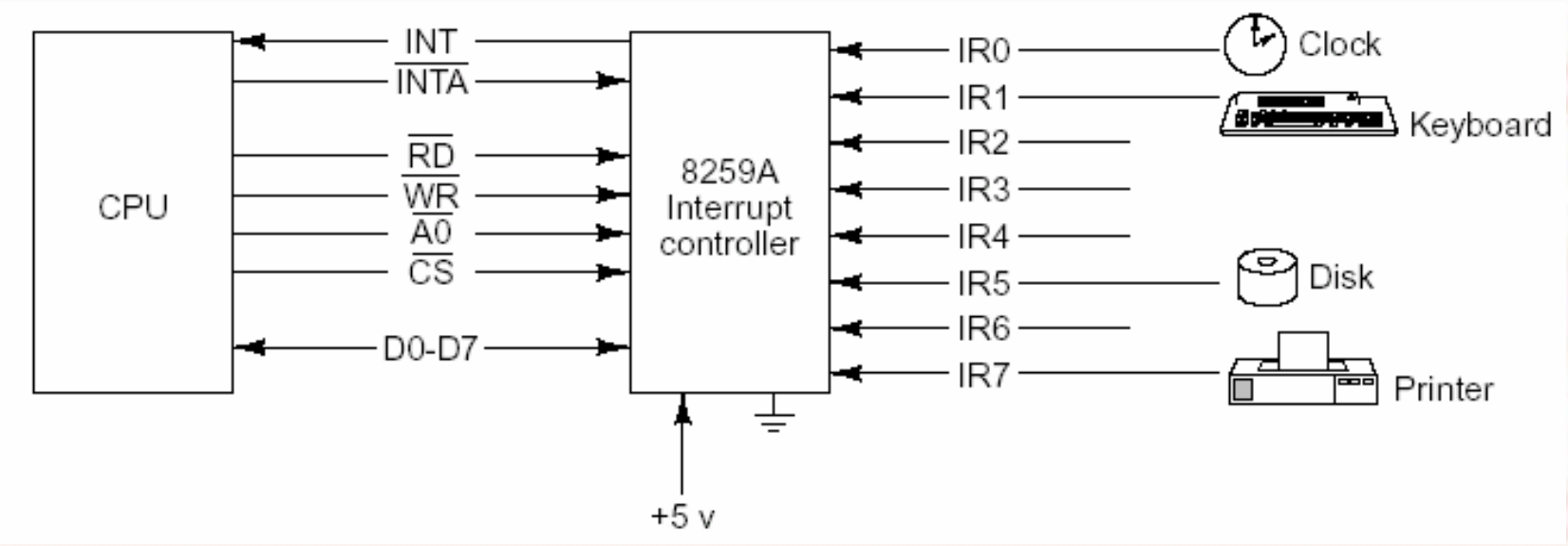



A typical modern PC with a PCI bus, and an ISA bus. The modem and sound card are ISA devices, the SCSI controller is a PCI device.

# Logical Structure of a Simple Personal Computer



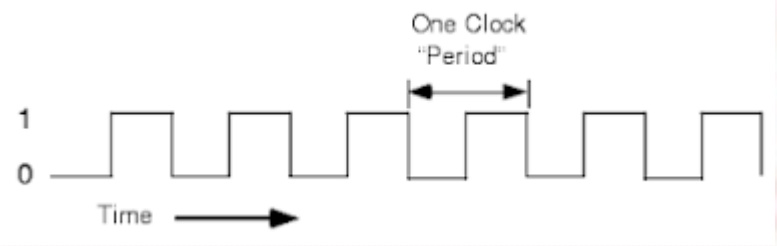
# 8259A interrupt controller



# Performance Measures

- Computer user
  - measures performance based on the time taken to execute a given program
  - The time required to execute a job by a computer is often expressed in terms of clock cycles
- **Laboratory engineer**
  - **considers the** total amount of work done in a given time

- If we use the following notation:
  - CC cycle count
    - number of clock cycles for executing a job
  - CT cycle time
  - f = 1/CT frequency




  - The time taken by the CPU to execute a job is:
    - CPU time = CC * CT = CC/f

- The average number of clock cycles per instruction (CPI)
  - an alternate performance measure:

$$CPI = \frac{CPU\ clock\ cycles\ for\ the\ program}{Instruction\ count}$$

$$CPU\ time = Instruction\ count \cdot CPI \cdot Clock\ cycle\ time =$$

$$= \frac{Instruction\ count \cdot CPI}{Clock\ rate}$$

  - the instruction set of a given machine consists of number of instruction categories: *ALU*, *load*, *store*, *branch*, and others

- In case that the CPI for each instruction category is known, the overall CPI can be calculated as:

$$CPI = \frac{\sum_{i=1}^{n} CPI_i \cdot I_i}{Instruction\ count}$$

  - where $I_i$ is the number of times an instruction of type i is executed in the program, and $CPI_i$ is the average number of clock cycles needed to execute such instruction

- CPI reflects the organization and the instruction set architecture of the processor
- the instruction count reflects the instruction set architecture and compiler technology used

- A different performance measure
  - the rate of instruction execution per unit time, MIPS (million instructions-per-second):

$$MIPS = \frac{Instruction\ count}{Execution\ time \cdot 10^6} = \frac{Clock\ rate}{CPI \cdot 10^6}$$

$$CPI = \frac{CPU\ clock\ cycles\ for\ the\ program}{Instruction\ count}$$

$$CPU\ time = Instruction\ count \cdot CPI \cdot Clock\ cycle\ time = \frac{Instruction\ count \cdot CPI}{Clock\ rate}$$

  - Example:
    - Computing the overall CPI for the machines A and B
      - the following performance measures were recorded when executing a set of benchmark programs

| Instruction category | A - % of occurrence | A $CPI_i$ | B - % of occurrence | B $CPI_i$ |
|---|---|---|---|---|
| ALU | 38 | 1 | 35 | 1 |
| Load & store | 15 | 3 | 30 | 2 |
| Branch | 42 | 4 | 15 | 3 |
| Others | 5 | 5 | 20 | 5 |

What are overall CPI and MIPS rating for each of the machines A and B assuming that the clock rate of both CPUs as 200 MHz?

Solution:

$CPI_a = 2.76, \quad MIPS_a = 72.46$
$CPI_b = 2.4, \quad MIPS_b = 83.33$

- be careful in using MIPS to compare machines that have different instruction sets
  - MIPS does not track execution time
  - **Example:**

| Instruction category | No.of instructions (in millions) | No. of cycles Per instruction |
|---|---|---|
| Machine A | | |
| ALU | 8 | 1 |
| Load&Store | 4 | 3 |
| Branch | 2 | 4 |
| Others | 4 | 3 |
| Machine B | | |
| ALU | 10 | 1 |
| Load&Store | 8 | 2 |
| Branch | 2 | 4 |
| Others | 4 | 3 |

$$CPI_a = \frac{\sum_{i=1}^{n} CPI_i \cdot I_i}{Instruction\ count} \cong 2.2, \quad MIPS_a = \frac{Clock\ rate}{CPI_a \cdot 10^6} = \frac{200 \cdot 10^6}{2.2 \cdot 10^6} \cong 90.9$$

$$CPU_a = \frac{Instruction\ count \cdot CPI_a}{Clock\ rate} = 0.198\,s$$

$$CPI_b = \frac{\sum_{i=1}^{n} CPI_i \cdot I_i}{Instruction\ count} = 2.1, \quad MIPS_b = \frac{Clock\ rate}{CPI_b \cdot 10^6} = \frac{200 \cdot 10^6}{2.1 \cdot 10^6} = 95.2$$

$$CPU_b = \frac{Instruction\ count \cdot CPI_a}{Clock\ rate} = 0.21s$$

$$MIPS_b > MIPS_a \quad and \quad CPU_b > CPU_a$$

- Although machine B has a higher MIPS than A
  - it requires longer CPU time to execute the same set of benchmark programs
- Million floating point instructions per second, MFLOP
  - has also been used as measure for machines' performance

$$MFLOPS = \frac{Number\ of\ flo.pt.operations\ in\ a\ program}{Execution\ time \cdot 10^6}$$

# Inclass exercise 2

A compiler designer is trying to decide between two code sequences for a particular machine
The hardware designers have supplied the following:
For a particular high-level language, the compiler writer is considering two sequences that require the following instruction counts:

| Instruction class | CPI of the instruction class |
|---|---|
| A | 2 |
| B | 5 |
| C | 6 |

| Code sequence | Instruction counts (in millions) | | |
|---|---|---|---|
| | A | B | C |
| 1 | 6 | 7 | 5 |
| 2 | 3 | 8 | 2 |

What is the CPI for each sequence?
Which code sequence is faster ?

# benchmark suites

- One of the best-known benchmark suites is SPEC suite, produced by the Standard Performance Evaluation Corporation
- https://www.spec.org/

- Many of the benchmark suites use the geometric rather than the arithmetic mean to average the results of the programs contained in the benchmark suite

- The reason for that is that a single extreme value has less of an impact on the geometric mean than on the arithmetic mean.

$$Arithmetic\ mean = \frac{1}{n}\sum_{i=1}^{n} Execution\ time_i$$

$$Geometric\ mean = \sqrt[n]{\prod_{i=1}^{n} Execution\ time_i}$$

# geometric mean VS arithmetic mean

- Test data: 2.1, 2.2, 2.3
  - Avg =(2.1+2.2+2.3)/3
    = 2.2
  - GeoMean=$(2.1*2.2*2.3)^{1/3}$
    =2.1984

# geometric mean VS arithmetic mean

- Test data: 2.1, 2.2, 2.3,6.5
  - Avg =(2.1+2.2+2.3+6.5)/4
    = 3.275
  - GeoMean=$(2.1*2.2*2.3*6.5)^{1/4}$
    =2.883

# Programs to Evaluate Processor Performance

- (Toy) Benchmarks
  - 10-100 line program
  - e.g.: sieve, puzzle, quicksort
- Synthetic Benchmarks
  - Attempt to match average frequencies of real workloads
  - e.g., Whetstone, dhrystone
- • Kernels
  - Time critical excerpts

# Benchmark 'Games'

- Different configurations used to run the same workload on two systems
- Compiler wired to optimize the workload
- Test specification written to be biased towards one machine
- Workload arbitrarily picked
- Very small benchmarks used
- Benchmarks manually translated to optimize performance

# Common Benchmark Mistakes

- Only average behaviour represented in test workload
- Skewness of device demands ignored
- Loading level controlled inappropriately
- Caching effects ignored
- Buffer sizes not appropriate
- Inaccuracies due to sampling ignored

- Ignoring monitoring overhead
- Not validating measurements
- Not ensuring same initial conditions
- Using device utilizations for performance comparisons
- Collecting too much data but doing too little analysis

# SPEC: System Performance Evaluation Cooperative

- First Round 1989
  - 10 programs yielding a single number
- Second Round 1992
  - SpecInt92 (6 integer programs) and SpecFP92 (14 floating point programs)
  - Compiler flags set differently for different programs
- Third Round 1995
  - Single flag setting for all programs
  - New set of programs
  - "Benchmarks useful for 3 years"

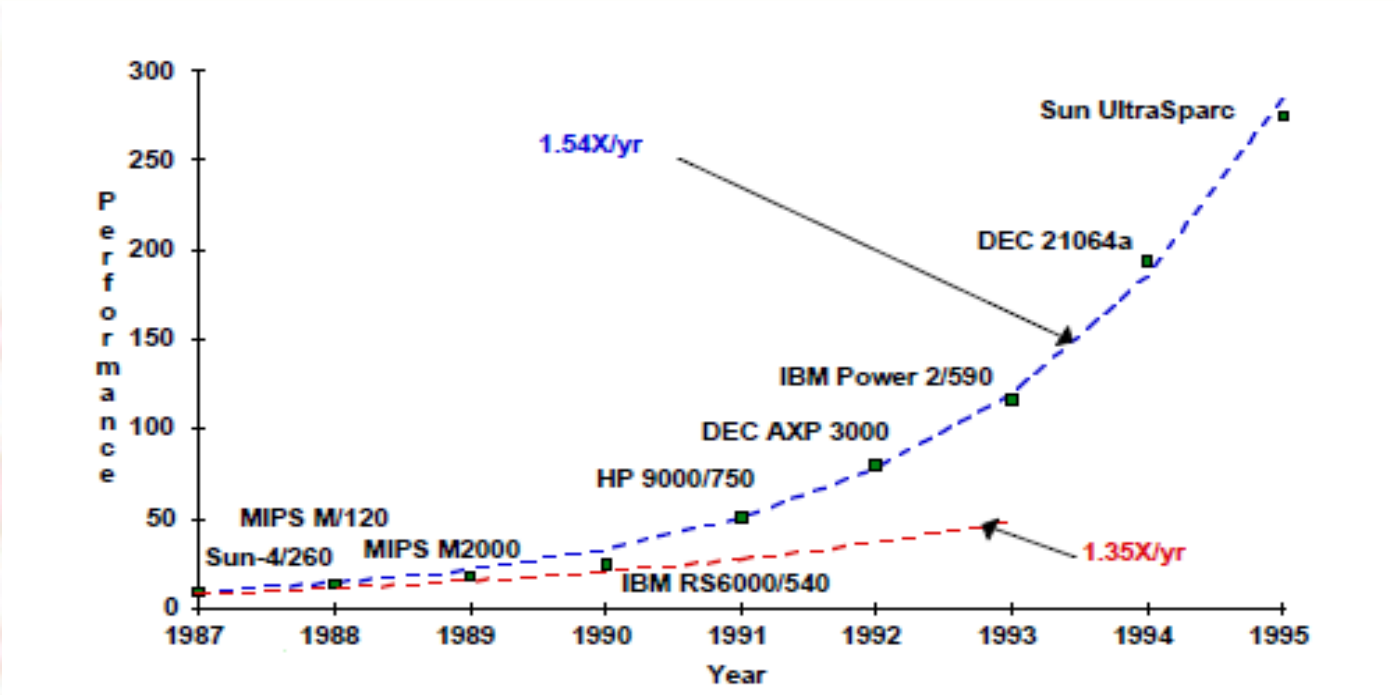
# Technology Improvement

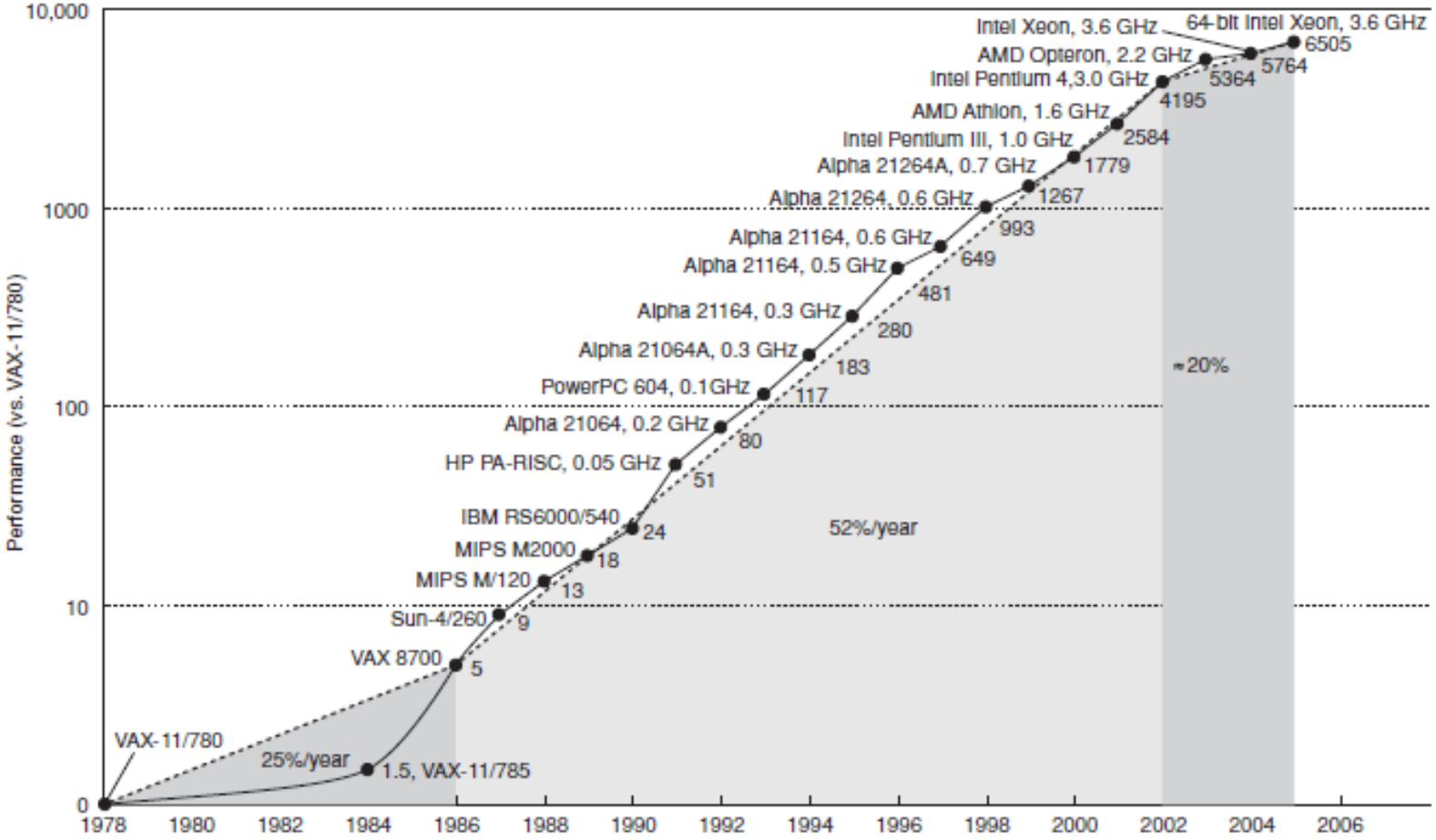
| | Capacity | Speed |
|---|---|---|
| Logic | 2x in 3 years | 2x in 3 years |
| DRAM | 4x in 3 years | 1.4x in 10 years |
| Disk | 2x in 3 years | 1.4x in 10 years |

**Speed increases of memory and I/O have not kept pace with processor speed increases.**

# Processor Performance

- Workstation performance improves roughly 50% per year
- Improvement in cost-performance ratio estimated at 70% per year

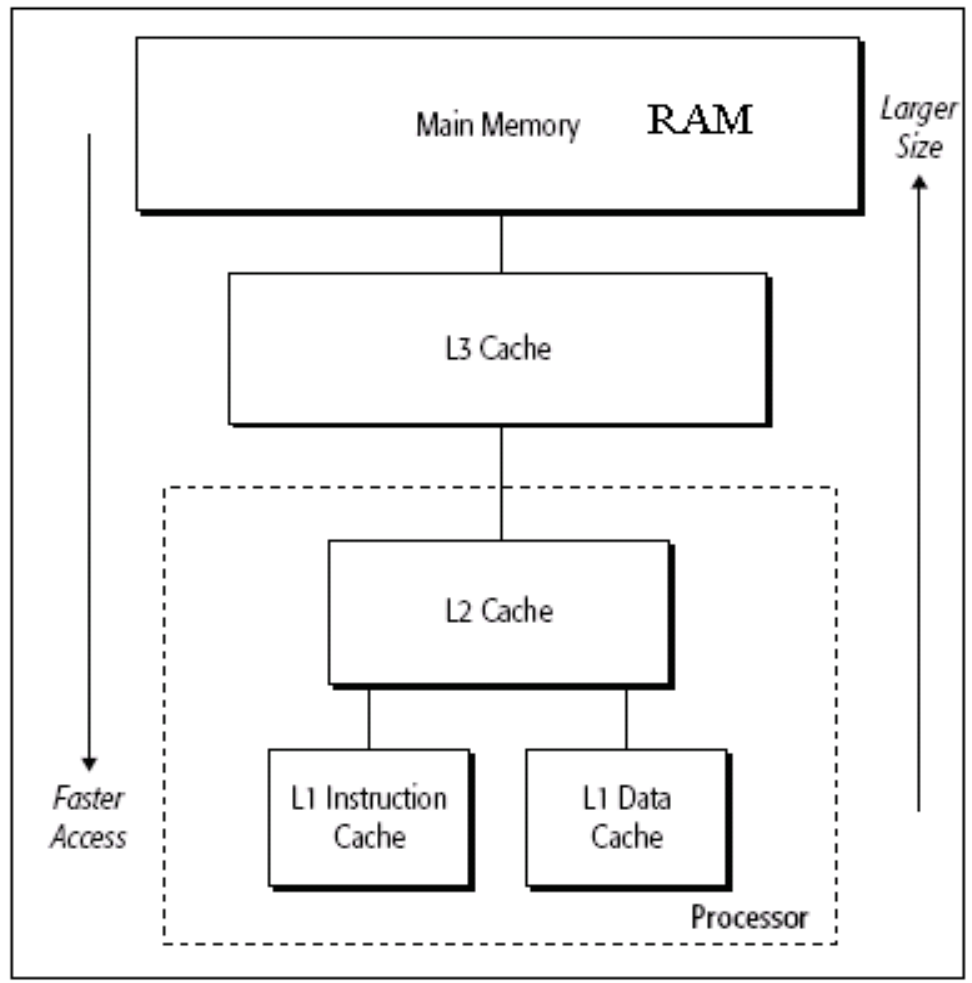



Performance (vs. VAX-11/780)
10,000
1000
100
10
0
1978
1980
1982
1984
1986
1988
1990
1992
1994
1996
1998
2000
2002
2004
2006
Intel Xeon, 3.6 GHz
64-bit Intel Xeon, 3.6 GHz
6505
AMD Opteron, 2.2 GHz
5764
Intel Pentium 4,3.0 GHz
5364
AMD Athlon, 1.6 GHz
4195
Intel Pentium III, 1.0 GHz
2584
Alpha 21264A, 0.7 GHz
1779
Alpha 21264, 0.6 GHz
1267
Alpha 21164, 0.6 GHz
993
Alpha 21164, 0.5 GHz
649
Alpha 21164, 0.3 GHz
481
Alpha 21064A, 0.3 GHz
280
PowerPC 604, 0.1GHz
183
Alpha 21064, 0.2 GHz
117
HP PA-RISC, 0.05 GHz
80
IBM RS6000/540
51
MIPS M2000
24
MIPS M/120
18
Sun-4/260
13
VAX 8700
9
VAX-11/780
5
1.5, VAX-11/785
25%/year
52%/year
≈20%

# Corollary: Make the Common Case Fast

- All instructions require an instruction fetch, only a fraction requires a data fetch/store
  - Optimize instruction access over data access
- Programs exhibit *locality*
  - 90% of time in 10% of code
  - Temporal Locality
  - Spatial Locality
- Access to small memories is faster
  - Provide a *storage hierarchy*
    - the most frequent accesses are to the closest memories

# Separate cache streams for data and instruction



# Principle of Locality

- Programs tend to reuse data and instructions they have used recently
  - A rule of thumb is that a program spends 90% of its execution time in only 10% of the code
  - ***Temporal locality***
    - recently accessed items are likely to be accessed in the near future
  - ***Spatial locality***
    - items whose addresses are near one another tend to be referenced close together in time

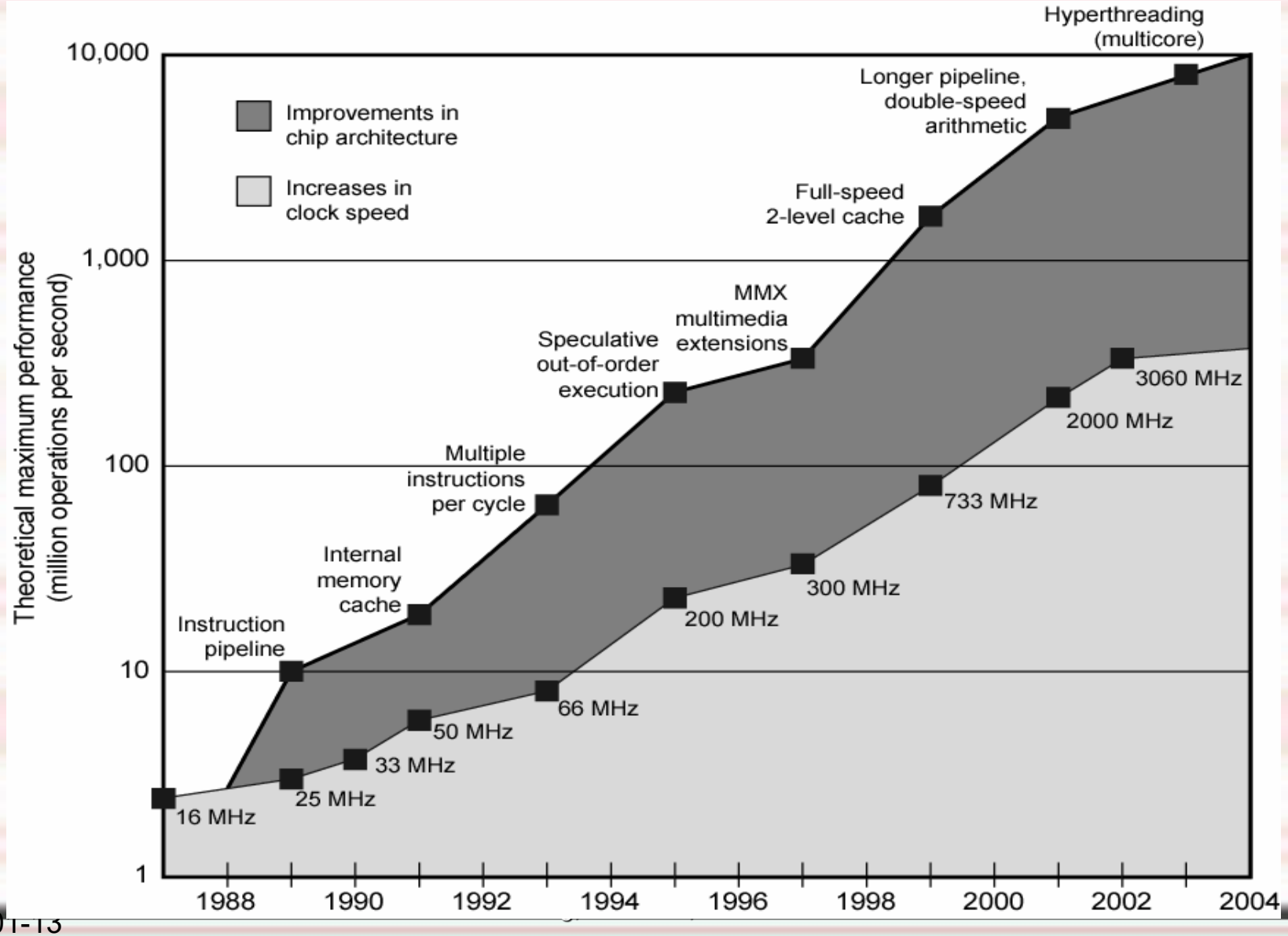
# Improvements in Chip Organization and Architecture

- Increase hardware speed of processor
  - Fundamentally due to shrinking logic gate size
  - More gates, packed more tightly, increasing clock rate
  - Propagation time for signals reduced
- Increase size and speed of caches
  - Dedicating part of processor chip
    - Cache access times drop significantly
- • Change processor organization and architecture
  - Increase effective speed of execution
  - Parallelism

# Problem with Clock Speed and Logic Density

- Power
  - Power consumption increases with density of logic and clock speed
  - Dissipating heat
- RC delay
  - Speed at which electrons flow is limited by resistance and capacitance of metal wires connecting them
  - Delay increases as RC product increases
  - Wire interconnects thinner, increasing resistance
  - Wires closer together, increasing capacitance
- Memory latency
  - Memory speeds lag processor speeds

# Intel Microprocessor Performance



# Increase Cache Capacity

- Typically two or three levels of cache between processor and main memory
- Chip density increased
  - More cache memory on chip
  - Faster cache access
- Pentium chip devoted about 10% of chip area to cache
- Pentium 4 devotes about 50%

# More Complex Execution Logic

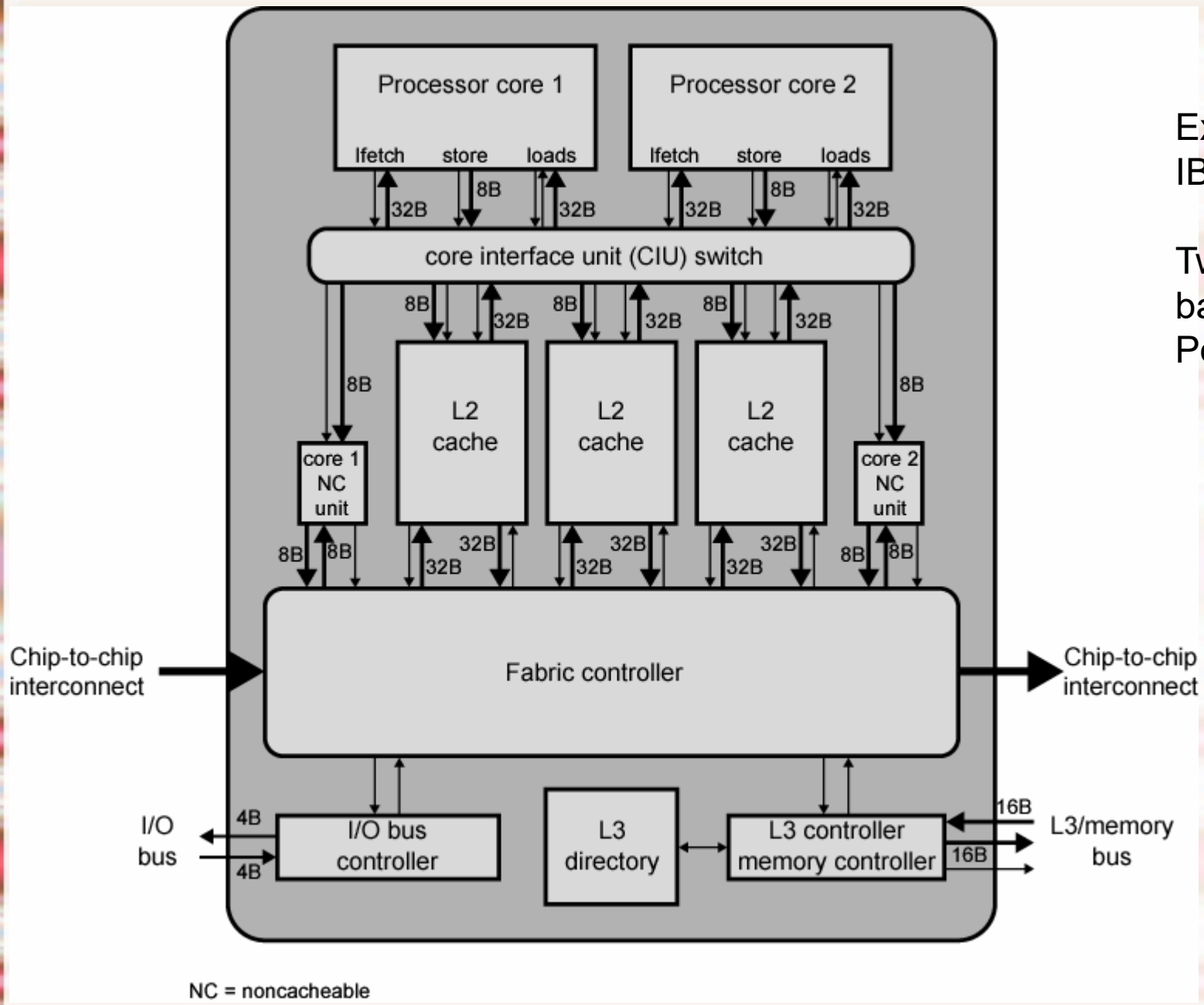
- Enable parallel execution of instructions
- Pipeline works like assembly line
  - Different stages of execution of different instructions at same time along pipeline
- Superscalar allows multiple pipelines within single processor
  - Instructions that do not depend on one another can be executed in parallel

# Diminishing Returns

- Internal organization of processors complex
  - Can get a great deal of parallelism
  - Further significant increases seam to be relatively modest
- Benefits from cache are reaching limit
- Increasing clock rate runs into power dissipation problem
  - Some fundamental physical limits are being reached

# New Approach – Multiple Cores

- Multiple processors on single chip
  - Large shared cache
- Within a processor, increase in performance proportional to square root of increase in complexity
- If software can use multiple processors, doubling number of processors almost doubles performance
- use two simpler processors on the chip rather than one more complex processor




Example:
IBM POWER4

Two cores based on PowerPC

W.Stallings "Computer Organization and Architecture", 7th edition, 2005

# Questions?

- ???