[Microprocessor Applications]

Introduction to Microprocessor

Chester Sungchung Park

SoC Design Lab, Konkuk University

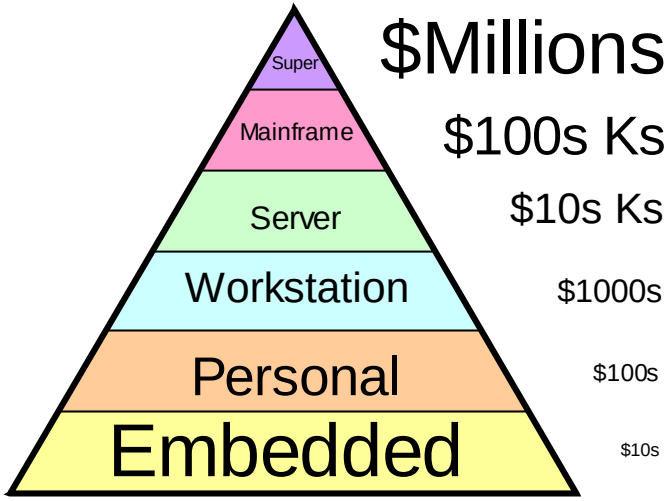
Webpage: http://soclab.konkuk.ac.kr

Outline

- Microprocessors
- Instruction Set Architecture (ISA)
- Microarchitecture

Computer

- What is a computer?
  - One that computes; specifically, a programmable electronic device that can store, retrieve and process data (Merriam-Webster Dictionary)
  - A machine that manipulates data according to a list of instructions (Wikipedia)
- Classification (power, price)
  - Supercomputers (e.g., Deep Blue)
  - Mainframes (e.g., IBM4381)
  - Personal computers
  - (Embedded) microcontrollers (e.g., AVR, PIC)

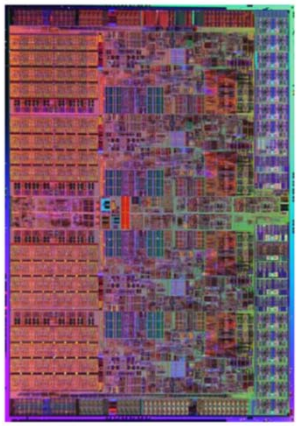


Personal Computers

- Any general-purpose computer intended to be operated directly by an end user
- Contains a processor called differently
  - Microprocessor: integrated on a single chip
  - Central Processing Unit (CPU)
  - Microprocessor Unit (MPU): similar to CPU

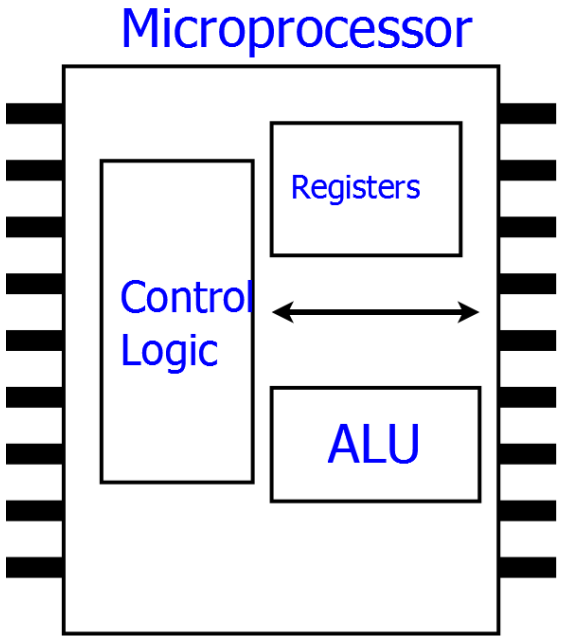
Microprocessor

- A computer processor which incorporates the functions of a computer's CPU on a single integrated circuit (IC) (Wikipedia)
- Consists of datapath, controller and buses



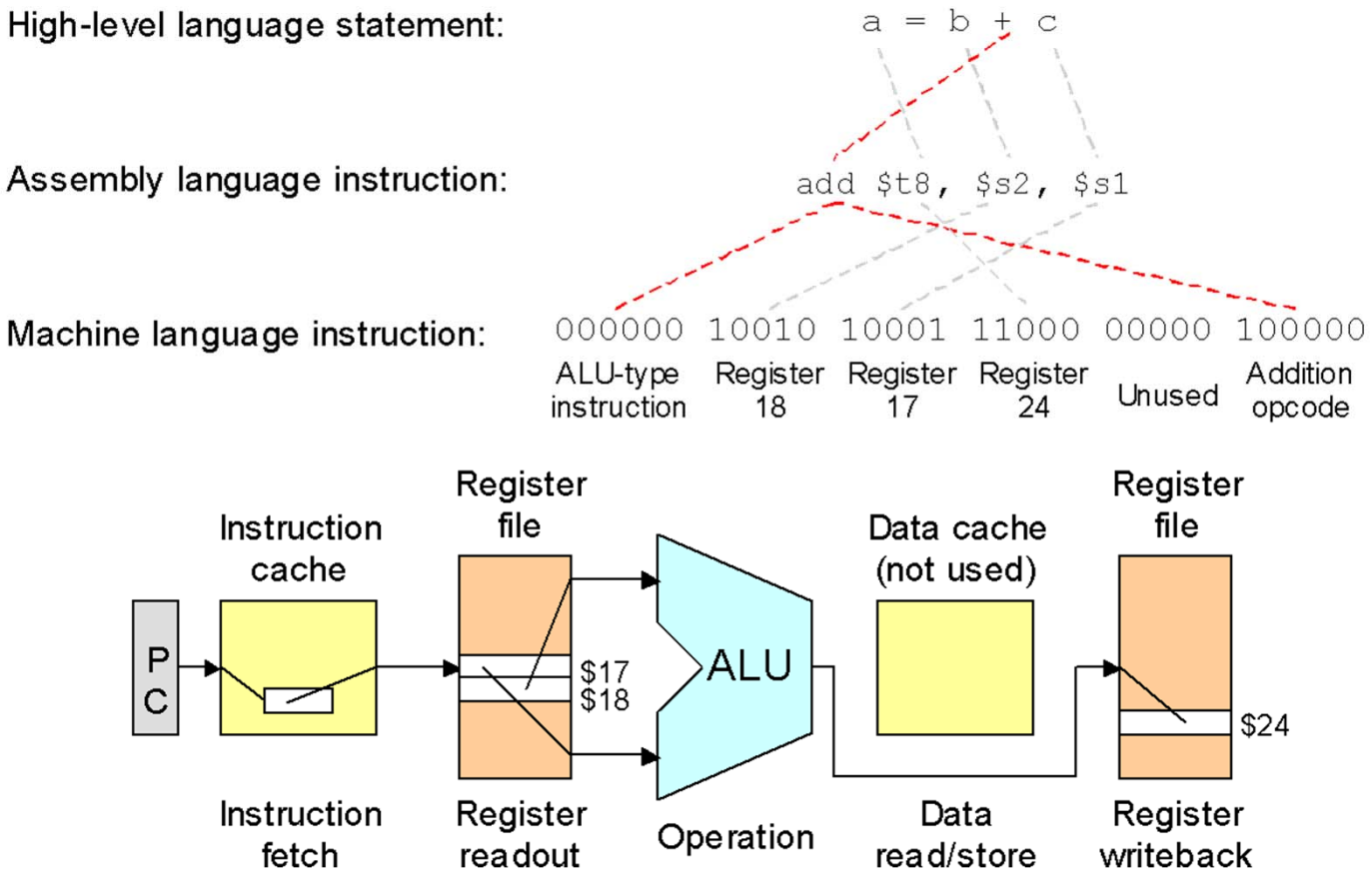
Core i7 die

Core i7 package



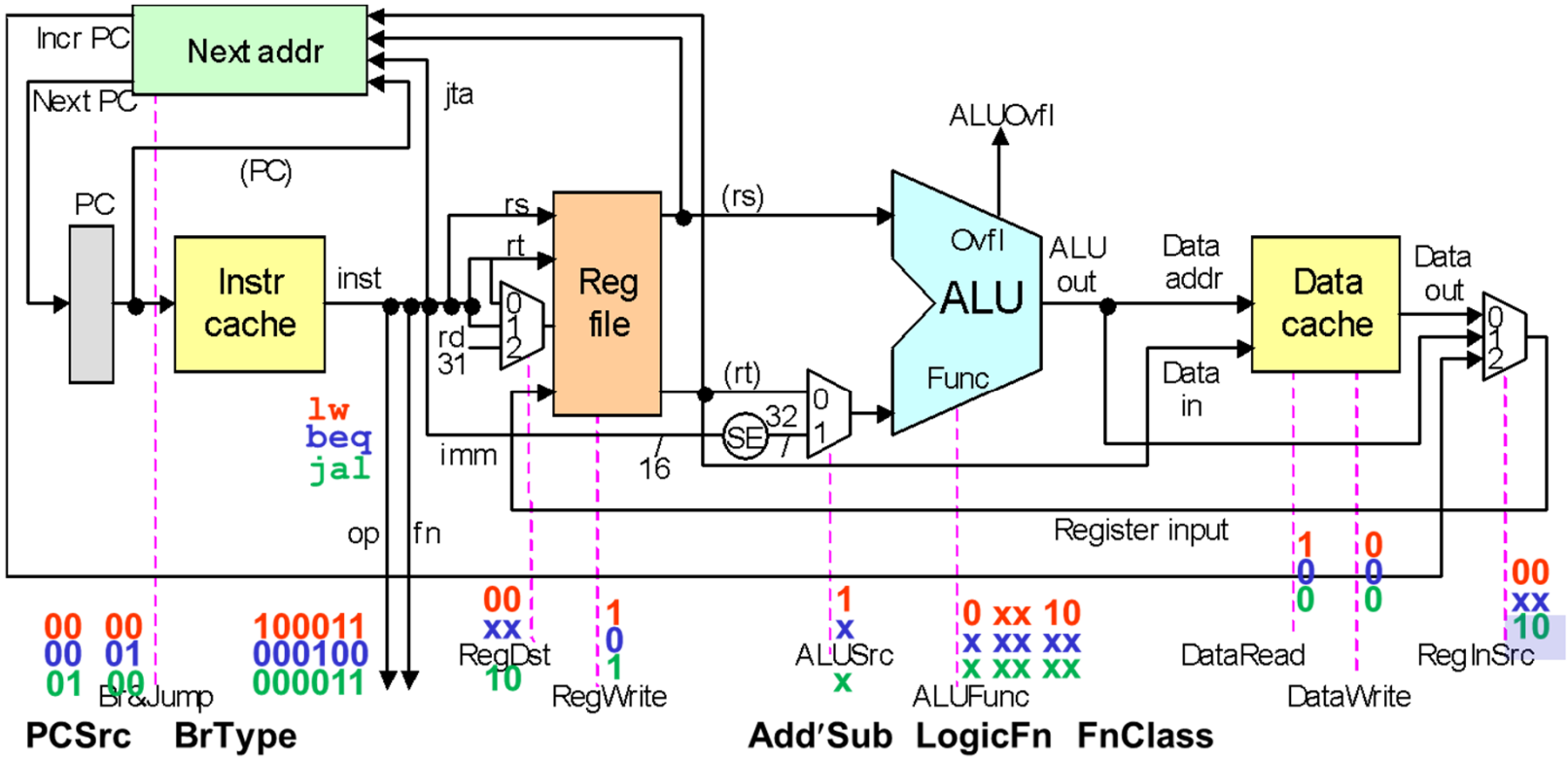
Microprocessor

- Datapath



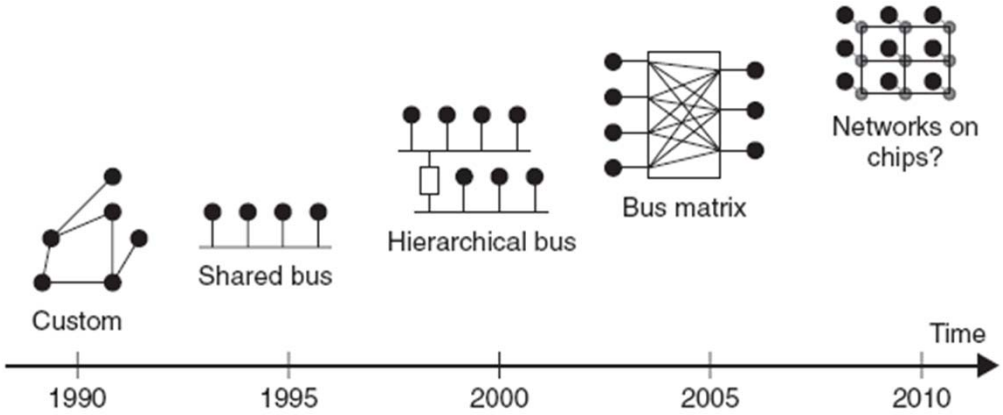
Microprocessor

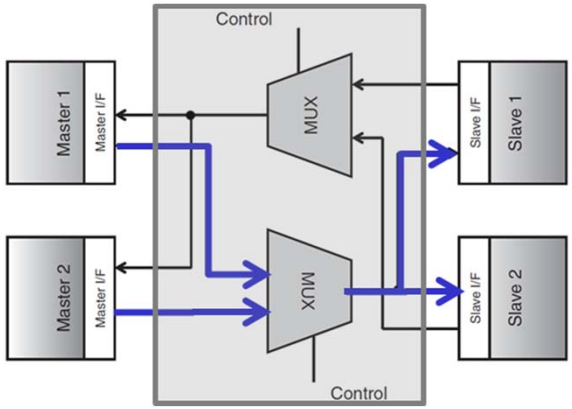
- Controller



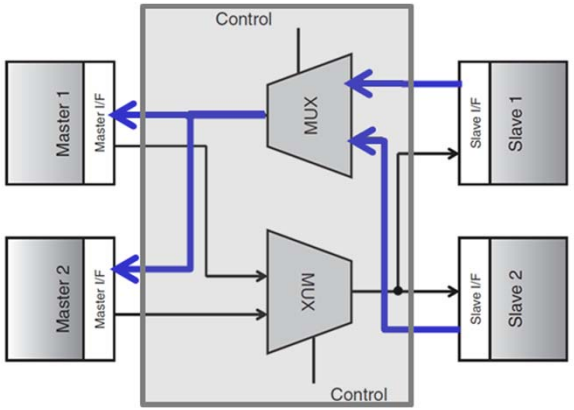
Microprocessor

- On-chip bus





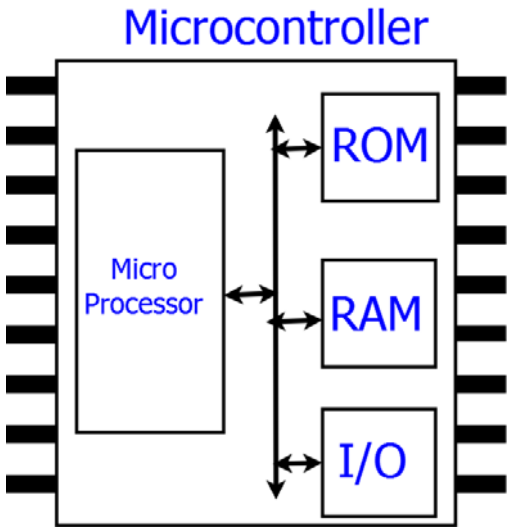
Master to Slave
(address, control, write data)

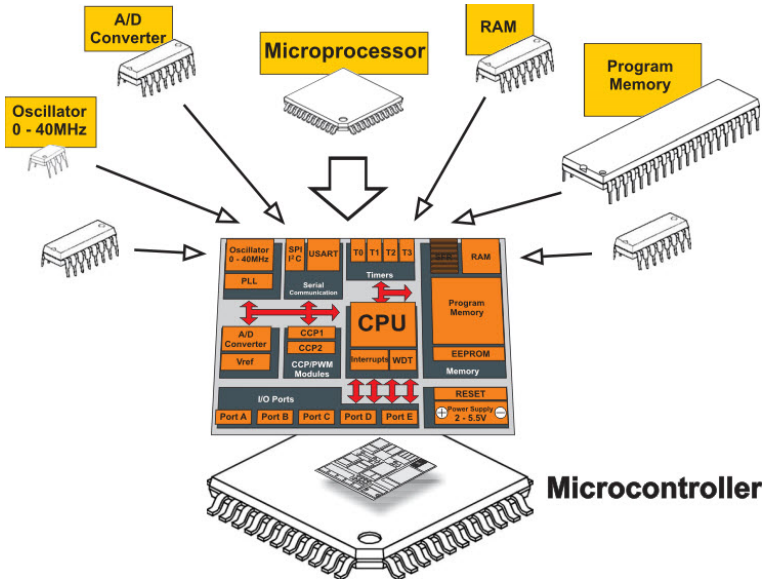


Slave to Master
(read data)

Microcontroller

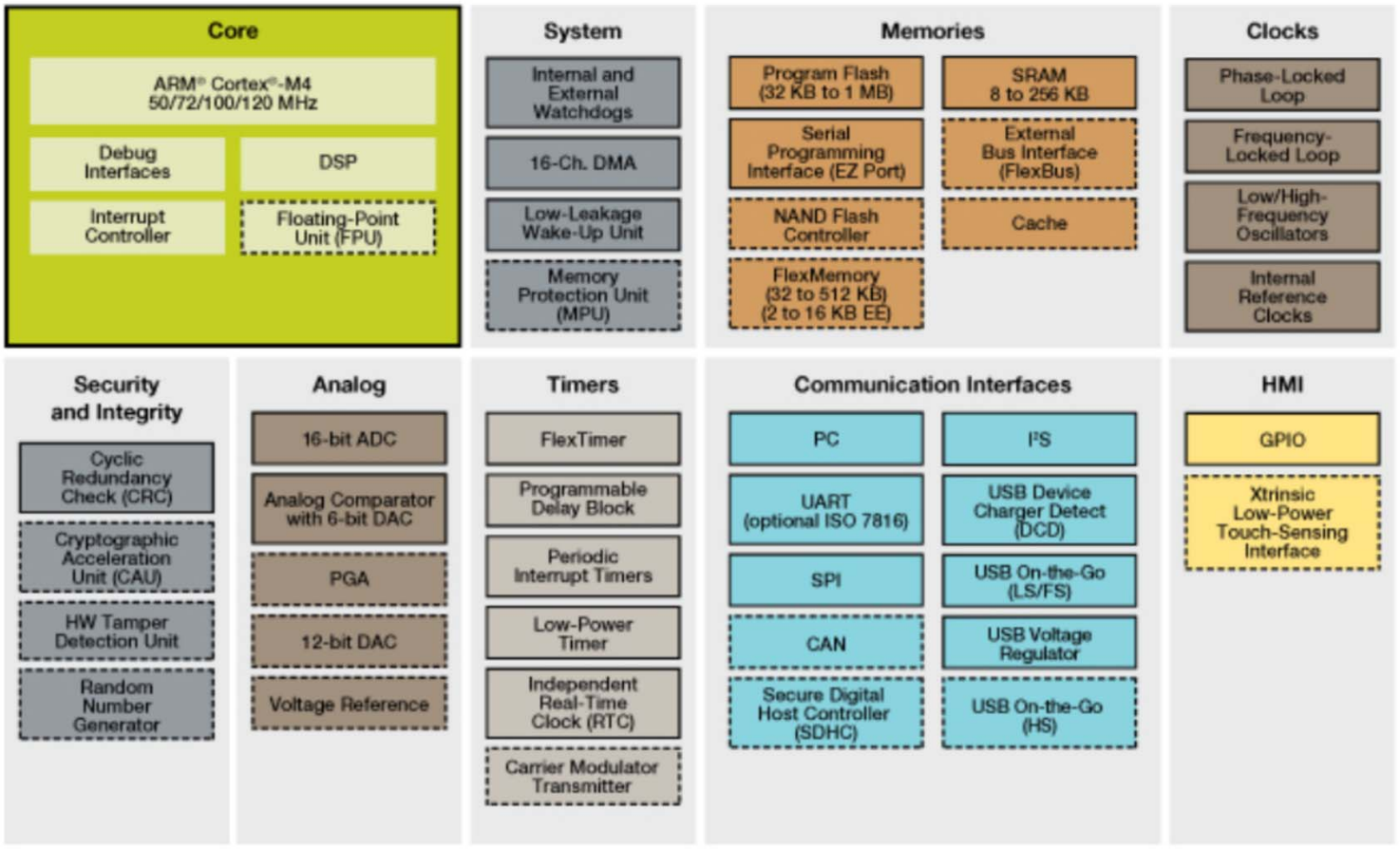
- Embedded system, a special-purpose computer system designed to perform one or a few dedicated functions (often with real-time)
  - Also called Microcontroller Unit (MCU)
- MCU = MPU + memory + IO peripherals
  - Plus support devices such as timers, A/D converters etc.





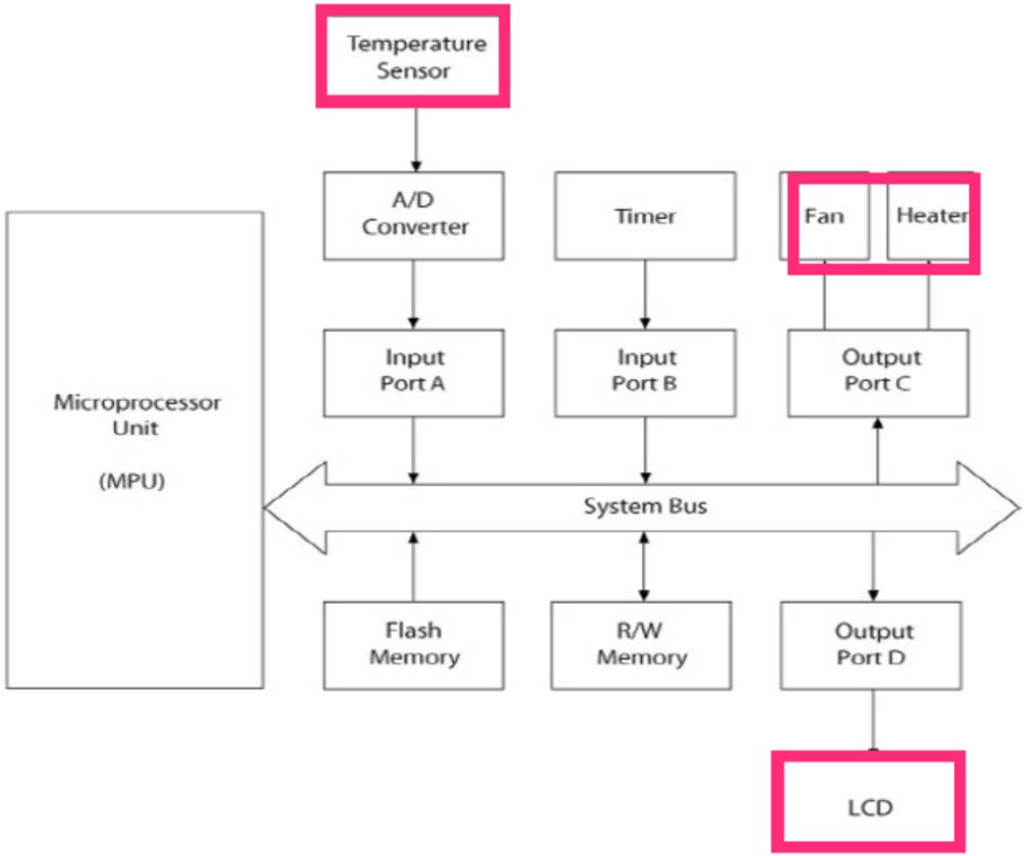
Microcontroller

- Example: Kinetis Kx2



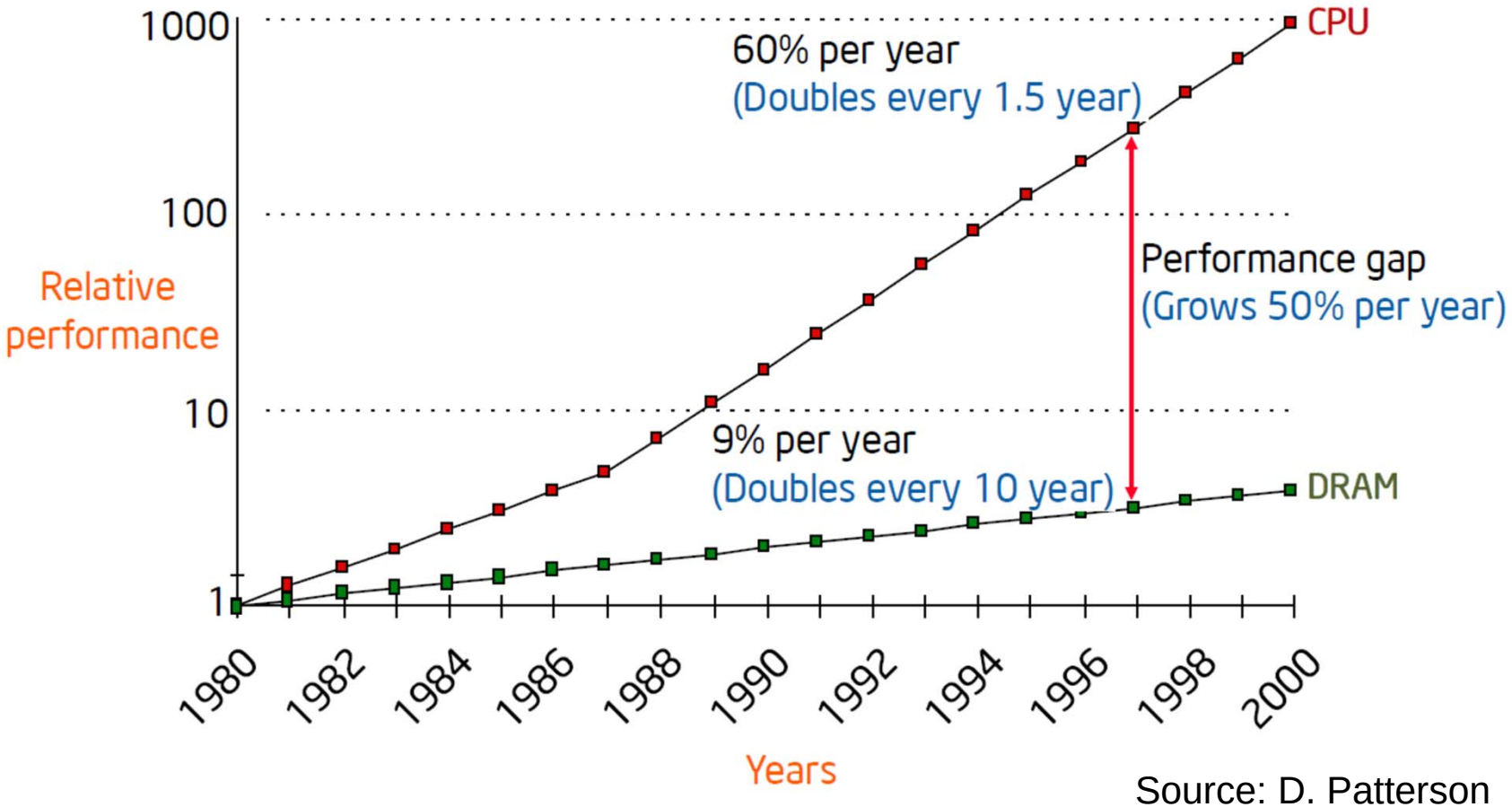
Microcontroller

- Example: temperature sensing system



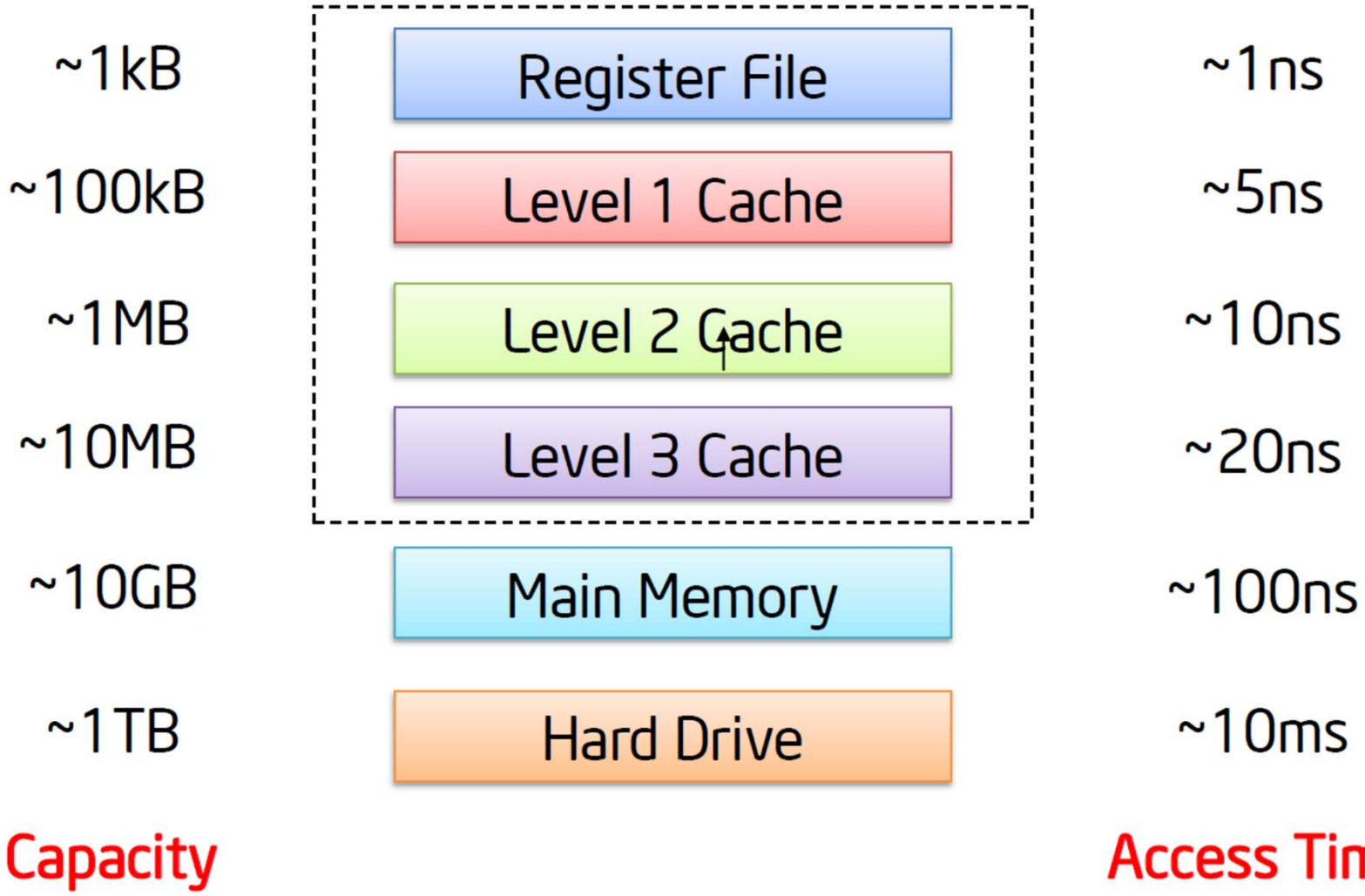
Memory Wall

- Performance gap between processor and memory



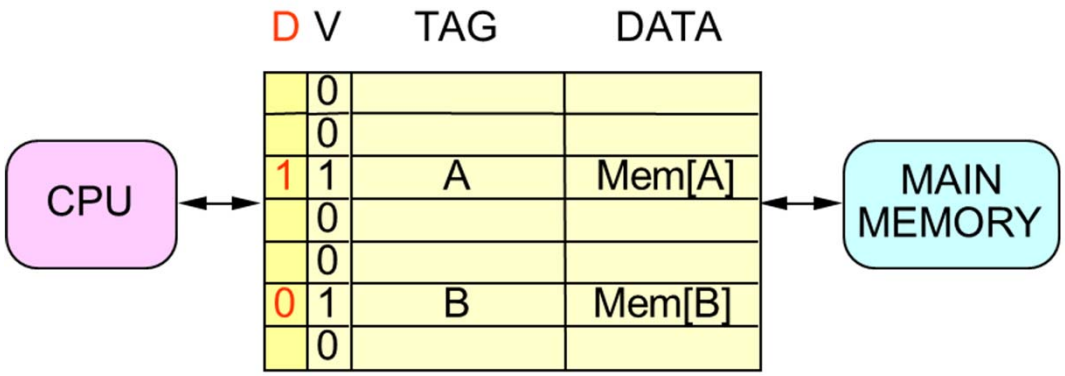
Source: D. Patterson

Memory Hierarchy



Cache Memories

- Cache with write-back

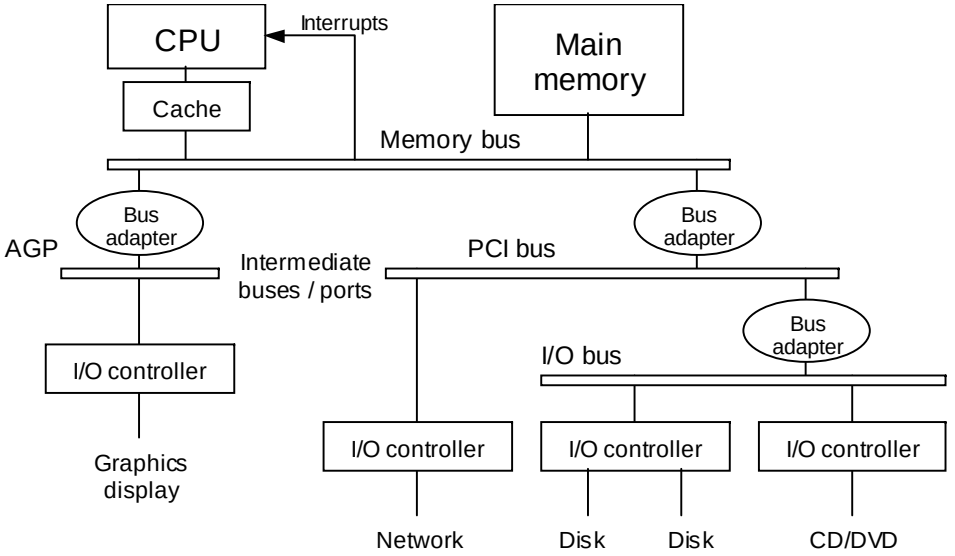


On reference to Mem[X]: Look for X among tags...

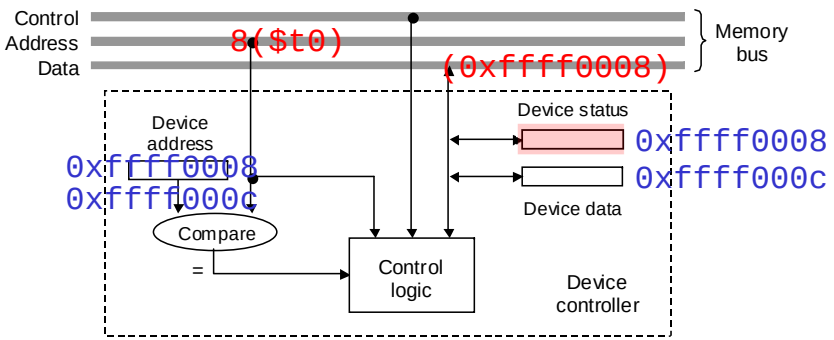
HIT: X == TAG(i) , *for some cache line* i
- **Read: return DATA(i)**
- **Write: change DATA(i); Start write to Mem[X] D(i)=1**

MISS: X *not found in* TAG *of any cache line*
- **Replacement selection:**
  - **Select some line k to hold Mem[X]**
  - **If D(k) == 1 (Write back) Write DATA(k) to Mem[TAG(k)]**
- **Read: Read Mem[X]; Set TAG(k) = X, DATA(k) = Mem[X] D(k)=0**
- **Write: Start write to Mem[X] D(k)=1**
  - **Set TAG(k) = X, DATA(k) = new Mem[X]**

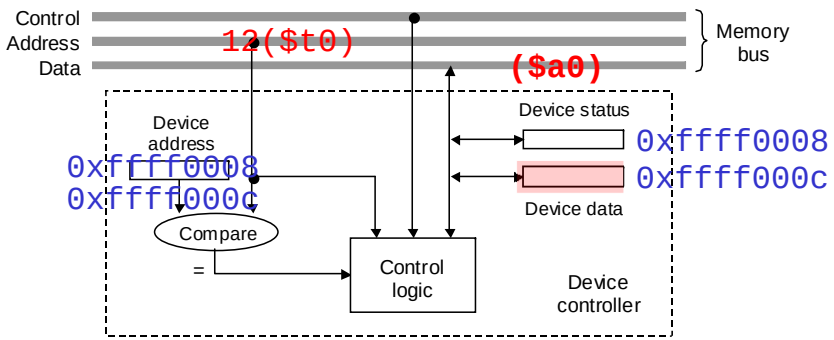
I/O Peripherals



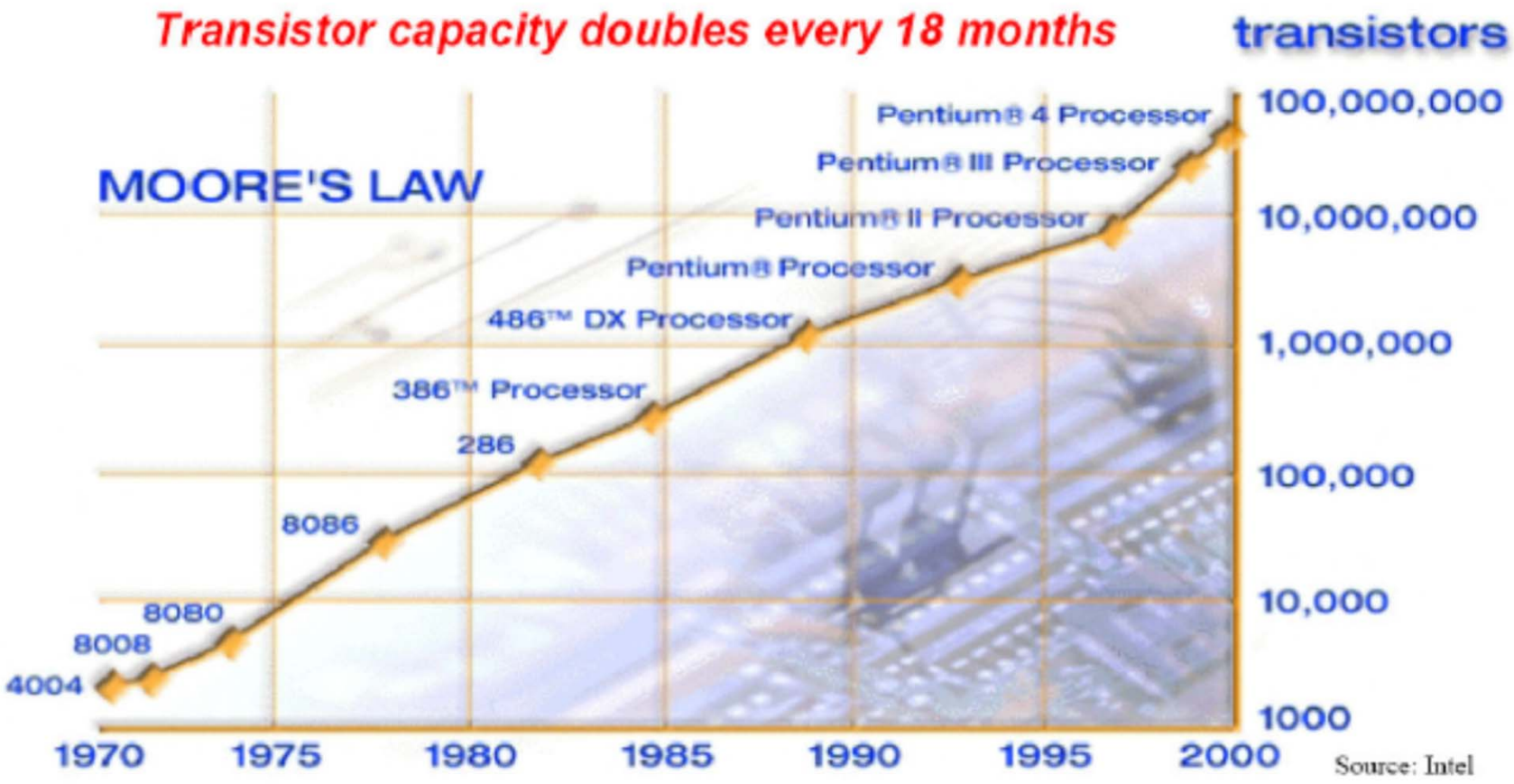

```
lw   $t1,8($t0)
```




```
sw   $a0,12($t0)
```



Evolution of Microprocessors



Source: Intel

Evolution of Microprocessors

Source: Intel

Instruction Set Architecture (ISA)

- Determines microprocessor architecture
  - E.g., IA32 (x86), IA64, ARM, PowerPC, SPARC, …
- Programs written for one processor can run on any other processor of the same ISA

Instruction	Usage
Load upper immediate	`lui rt,imm`
Add	`add rd,rs,rt`
Subtract	`sub rd,rs,rt`
Set less than	`slt rd,rs,rt`
Add immediate	`addi rt,rs,imm`
Set less than immediate	`slti rt,rs,imm`
AND	`and rd,rs,rt`
OR	`or rd,rs,rt`
XOR	`xor rd,rs,rt`
NOR	`nor rd,rs,rt`
AND immediate	`andi rt,rs,imm`
OR immediate	`ori rt,rs,imm`
XOR immediate	`xori rt,rs,imm`
Load word	`lw rt,imm(rs)`
Store word	`sw rt,imm(rs)`
Jump	`j L`
Jump register	`jr rs`
Branch less than 0	`bltz rs,L`
Branch equal	`beq rs,rt,L`
Branch not equal	`bne rs,rt,L`

Instruction	Usage
Move from Hi	`mfhi rd`
Move from Lo	`mflo rd`
Add unsigned	`addu rd,rs,rt`
Subtract unsigned	`subu rd,rs,rt`
Multiply	`mult rs,rt`
Multiply unsigned	`multu rs,rt`
Divide	`div rs,rt`
Divide unsigned	`divu rs,rt`
Add immediate unsigned	`addiu rs,rt,imm`
Shift left logical	`sll rd,rt,sh`
Shift right logical	`srl rd,rt,sh`
Shift right arithmetic	`sra rd,rt,sh`
Shift left logical variable	`sllv rd,rt,rs`
Shift right logical variable	`srlv rd,rt,rs`
Shift right arith variable	`srav rd,rt,rs`
Load byte	`lb rt,imm(rs)`
Load byte unsigned	`lbu rt,imm(rs)`
Store byte	`sb rt,imm(rs)`
Jump and link	`jal L`
System call	`syscall`

MiniMIPS instruction set

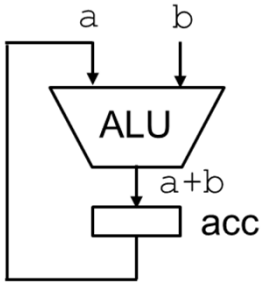
Instruction Set Architecture (ISA)

Example: Evaluating the expression (a + b) × (c – d)

```
push     a
push     b
add
push     d
push       c
subtract
multiply
```

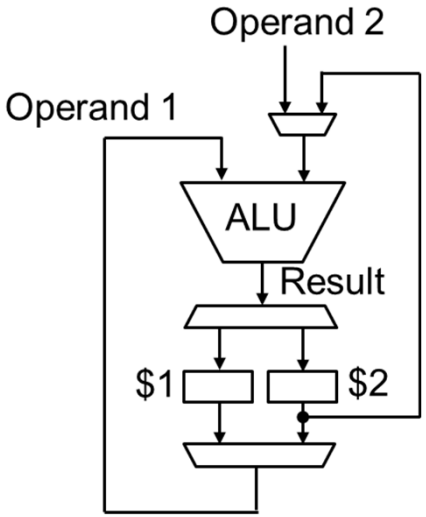
Zero-address

```
load     a
add      b
store    t
load     c
subtract d
multiply t
```



One-address

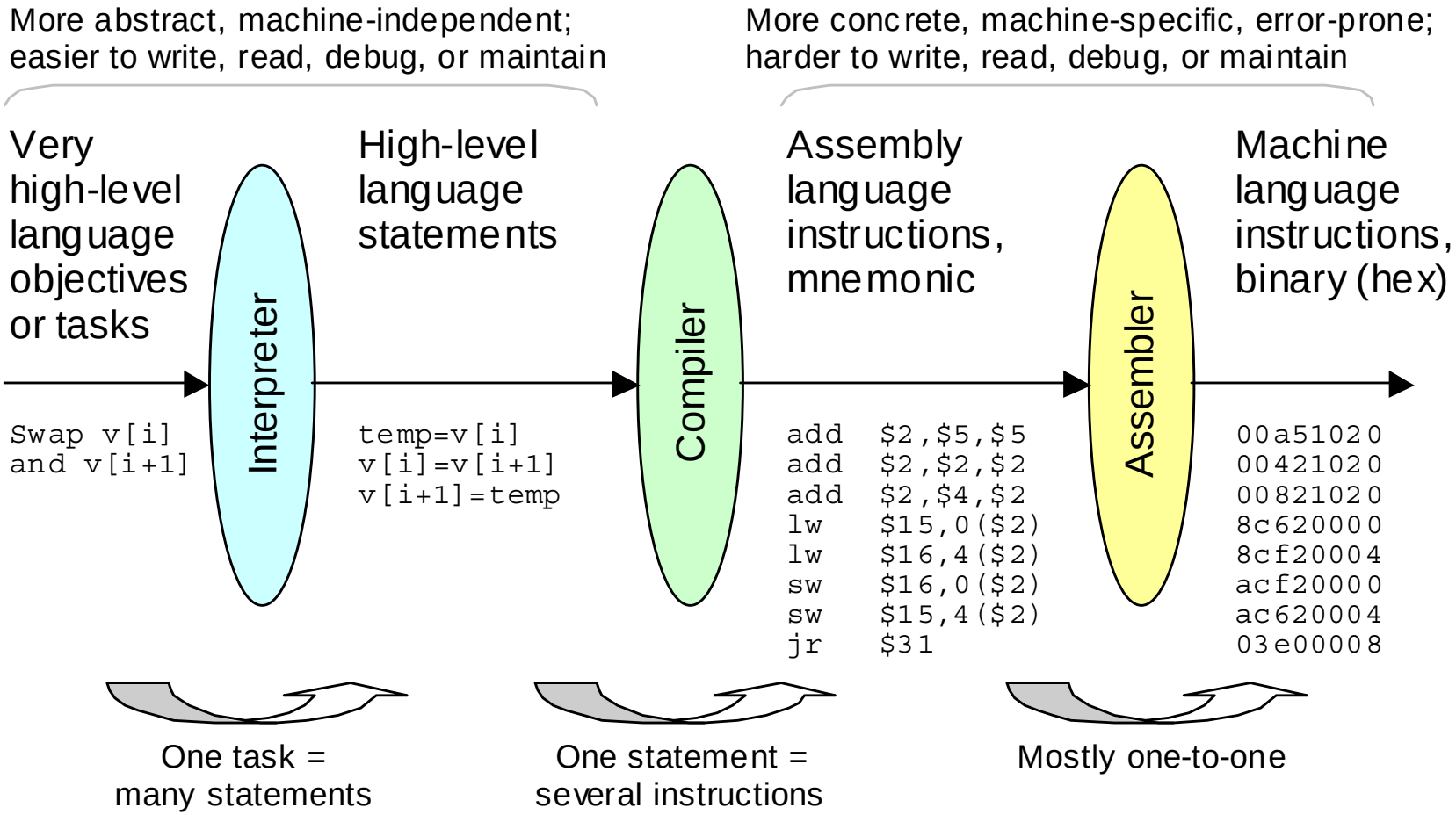
```
load     $1,a
add      $1,b
load     $2,c
subtract $2,d
multiply $1,$2
```



Two-address

Instruction Set Architecture (ISA)

- High-level vs. low-level language



Instruction Set Architecture (ISA)

- Example: MiniMIPS

High-level language program

```
Amax = A[0]
for(i = 1; i ≠ 4; i++)
  if(Amax < A[i])
    Amax = A[i];
```

Low-level language program

```
        lw    $t0,0($s1)
        addi  $t1,$zero,0
loop:   add   $t1,$t1,1
        beq   $t1,$s2,done
        add   $t2,$t1,$t1
        add   $t2,$t2,$t2
        add   $t2,$t2,$s1
        lw    $t3,0($t2)
        slt   $t4,$t0,$t3
        beq   $t4,$zero,loop
        addi  $t0,$t3,0
        j     loop
done:   ...
```

Memory

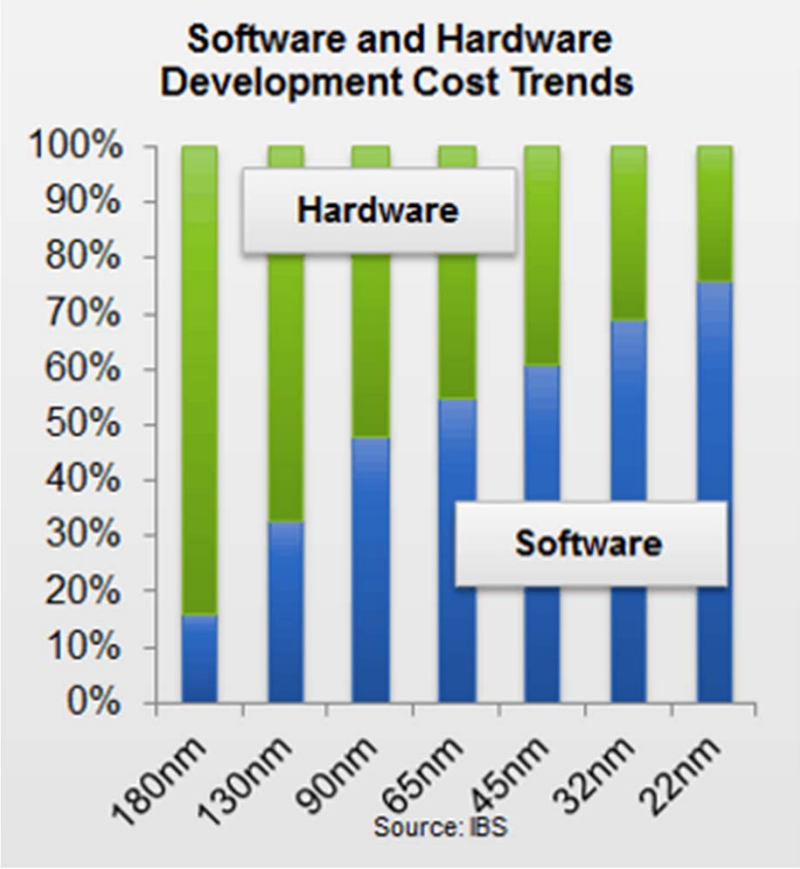
A	A[0]
A+4	A[1]
A+8	A[2]
A+12	A[3]

Registers

$t0	Amax
$t1	i
$t2	A+4i
$t3	A[i]
$t4	Amax<A[i]
$s1	A
$s2	4

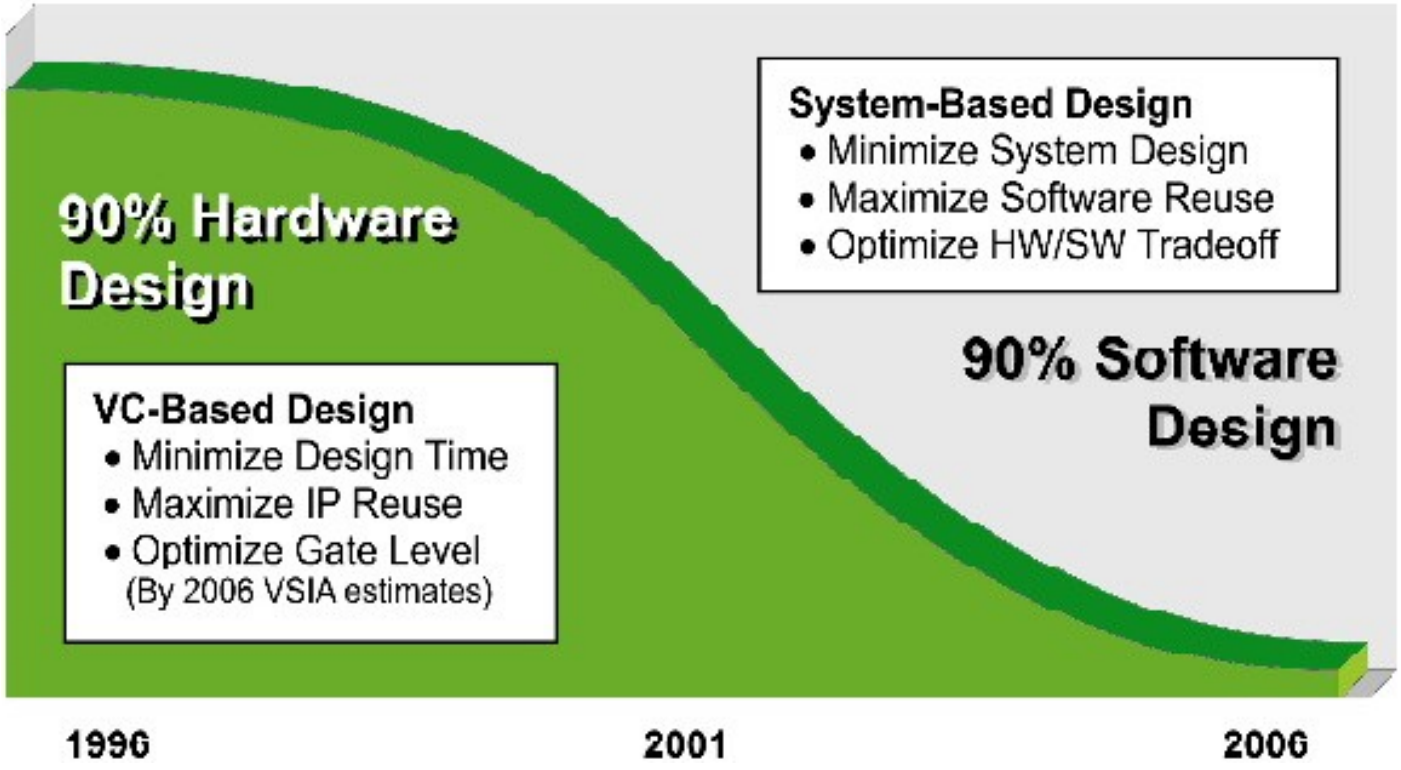
Importance of S/W

- Spending on S/W outpacing H/W



Importance of S/W

- Migration into S/W-centric design



Source: EETimes

Microarchitecture

- A set of steps that a processor takes to execute a particular ISA
- Microarchitecture features
  - Cache memory
  - Pipelining
  - Out-of-Order execution
  - Superscalar issue

Microarchitecture

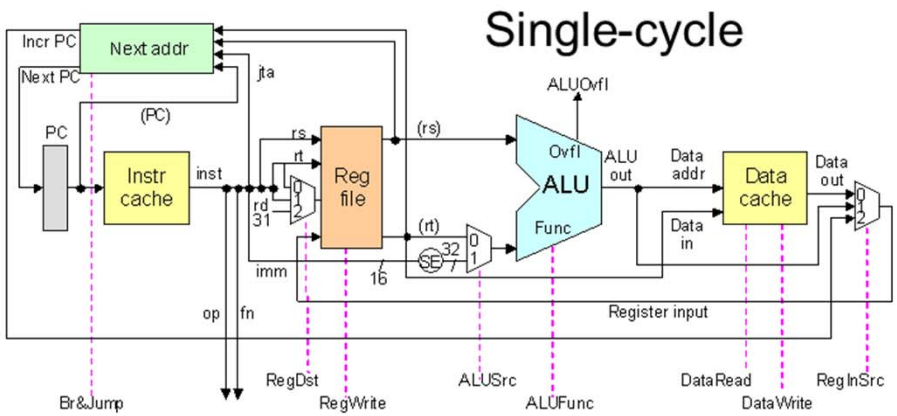
- w/o pipelining

Cycle	1	2	3	4	5	6	7	8	9
$Instr_1$	Fetch	Decode	Execute	Write					
$Instr_2$					Fetch	Decode	Execute	Write	
$Instr_3$									Fetch

- w/ pipelining

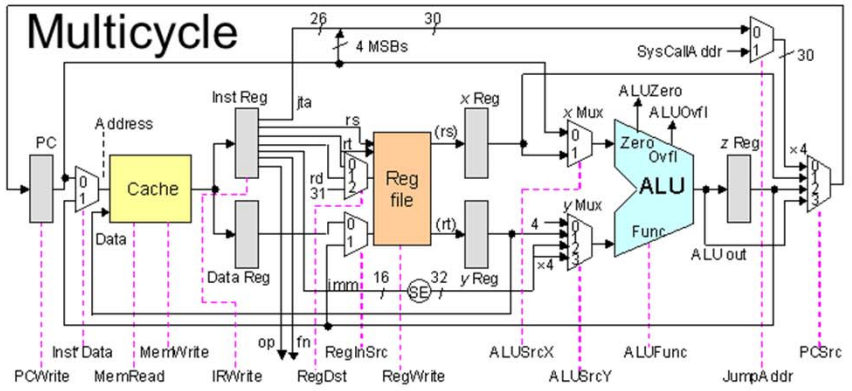
Cycle	1	2	3	4	5	6	7	8	9
$Instr_1$	Fetch	Decode	Execute	Write					
$Instr_2$		Fetch	Decode	Execute	Write				
$Instr_3$			Fetch	Decode	Execute	Write			
$Instr_4$				Fetch	Decode	Execute	Write		
$Instr_5$					Fetch	Decode	Execute	Write	
$Instr_6$						Fetch	Decode	Execute	Write

Microarchitecture



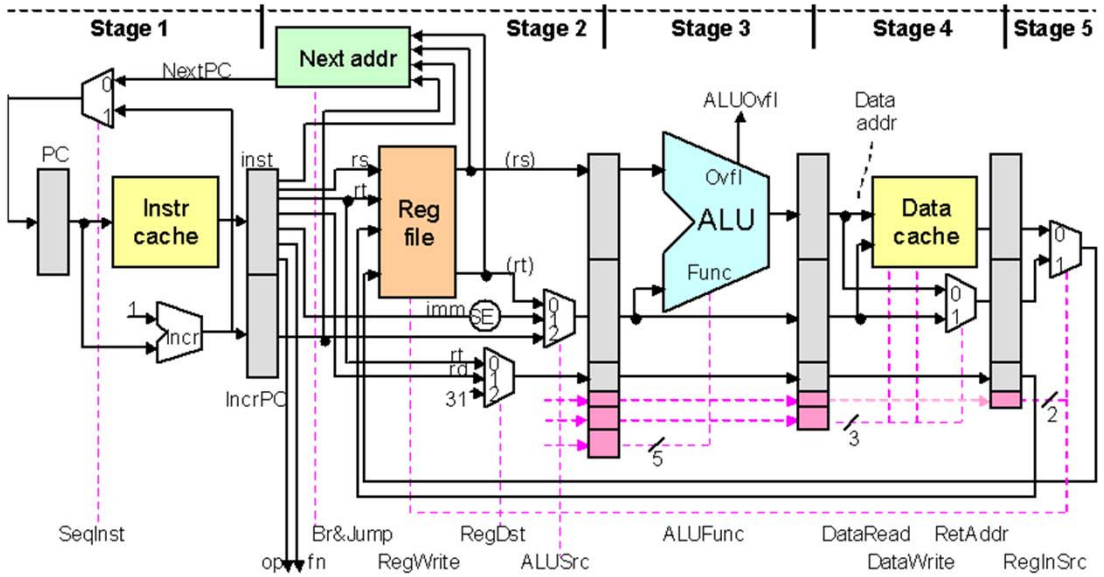
125 MHz

CPI = 1



500 MHz

CPI ≅ 4



Pipelined

500 MHz

CPI ≅ 1.1

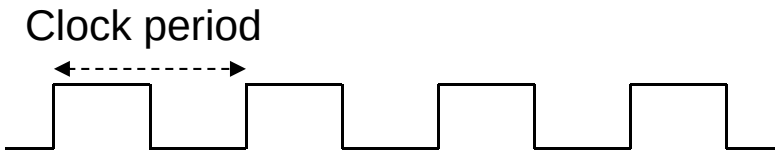
Performance of Microarchitecture

CPU time = Instructions × (Cycles per instruction) × (Secs per cycle)
= Instructions × Average CPI / (Clock rate)

Instructions: Number of instructions executed, not number of instructions in our program (dynamic count)

Average CPI: Is calculated based on the dynamic instruction mix and knowledge of how many clock cycles are needed to execute various instructions (or instruction classes)

Clock rate: 1 GHz = 10^9 cycles / s (cycle time 10^{-9} s = 1 ns)
200 MHz = 200 × 10^6 cycles / s (cycle time = 5 ns)



Performance of Microarchitecture

- CPU time affected by microarchitecture
  - More pipeline stages
    - ✓Less work in each cycle means better clock rate
    - ✓More dependencies means worse CPI
  - Superscalar issue & out-of-order execution
    - ✓Parallel work means better CPI
    - ✓More complexity can mean worse clock rate
- Whether CPU time improves or not depends on the **trade-off** between CPI and clock rate

References

- Computer system srchitecture, MIT OCW 6.823, Arvind and J. Emer
- Introduction to microprocessors, Intel Lab, Y. Baida
- Computer architecture – from microprocessors to supercomputer, Oxford University Press, B. Parhami