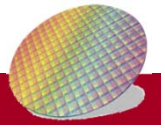
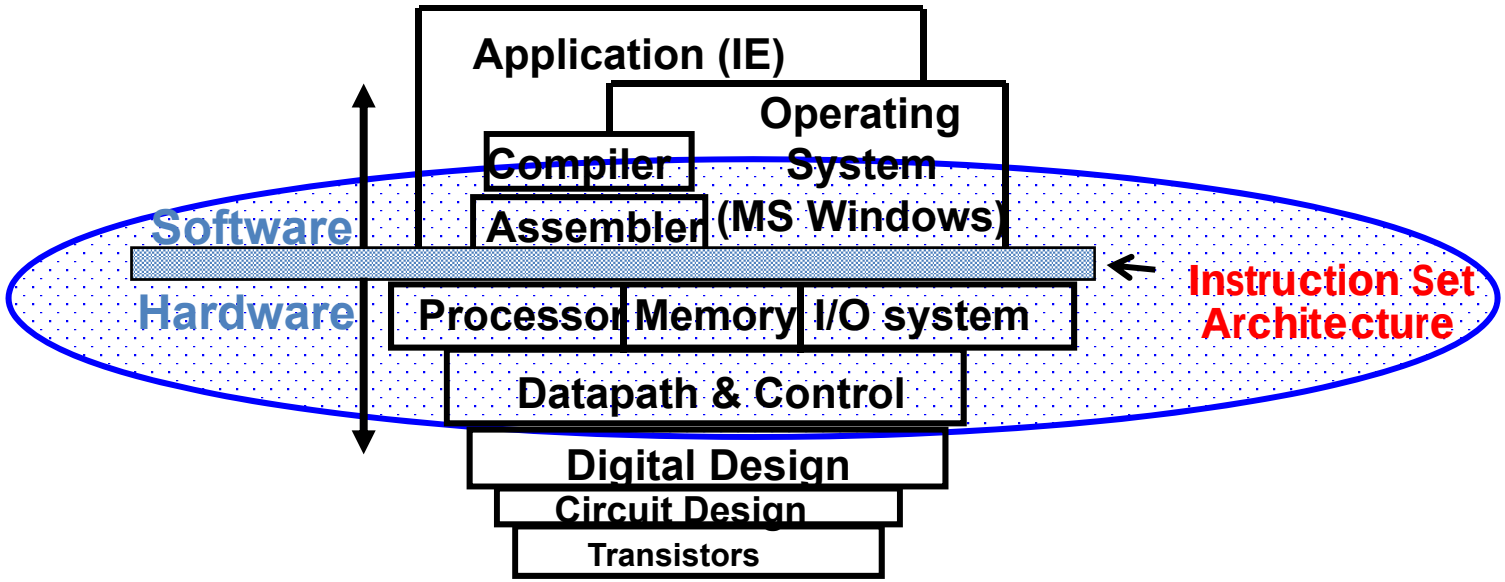
# Instruction Set Principle



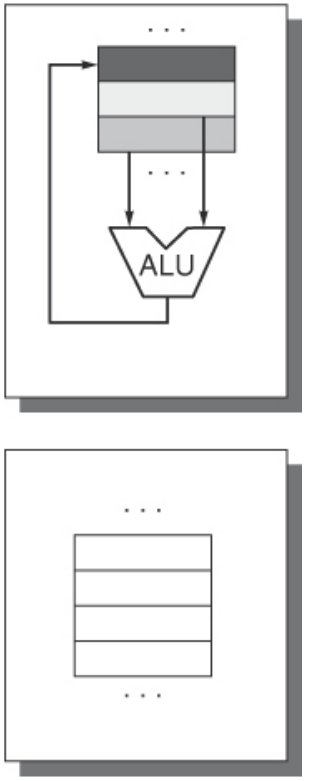
# Instruction Set

- Instruction Set : set of instructions of a computer
- Different computers have different instruction sets
  - But with many aspects in common
- Interface between hardware and software of a computer



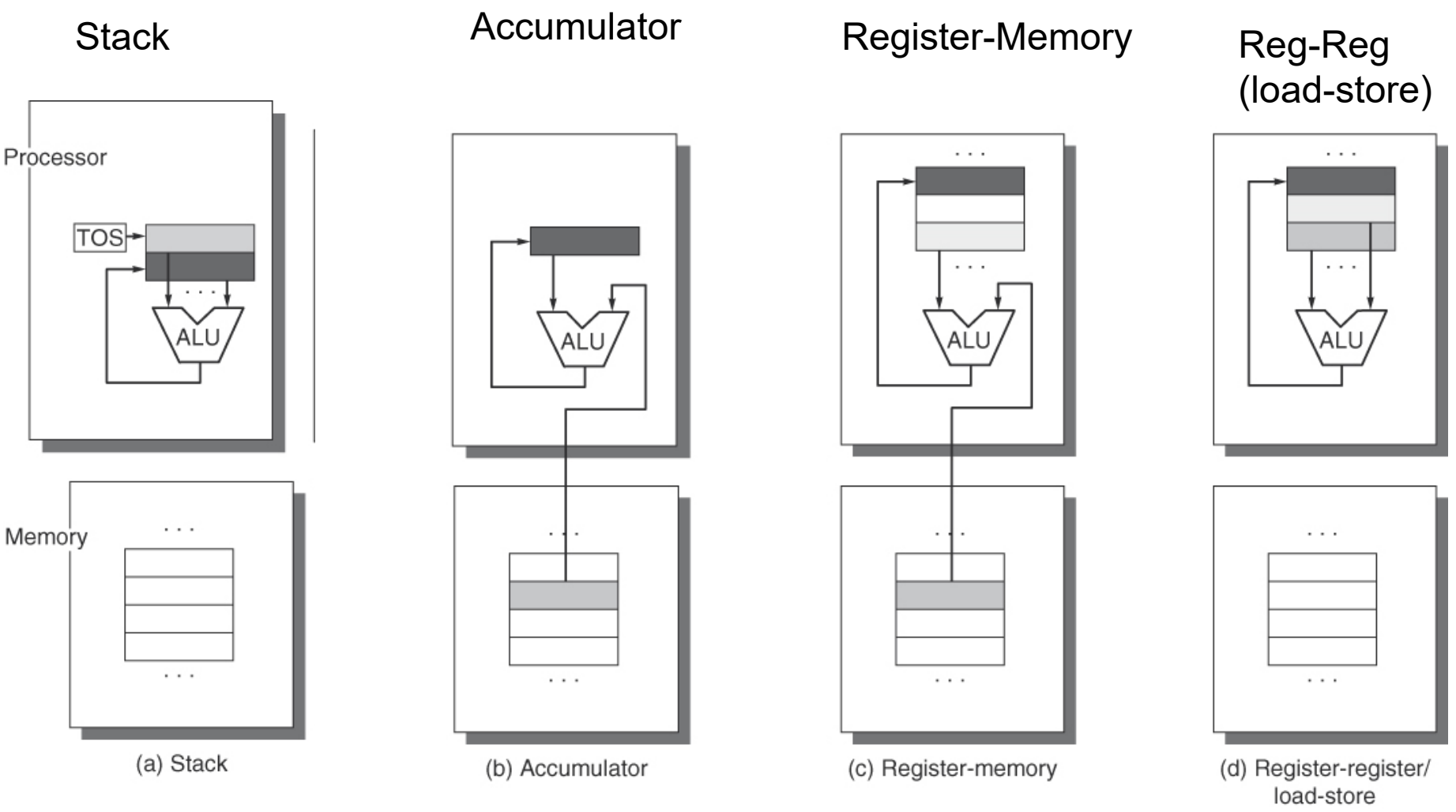
# Classifying Instruction Set Architecture

- Operands in instructions can be named explicitly or implicitly
- Types of Instruction Set Architecture
  - Register-register (load store): access memory only with load store instructions
  - Register-memory: access memory as part of any instruction
  - Stack: one implicit operand is the top of stack (TOS)
  - Accumulator: one implicit operand is accumulator




(d) Register-register/ load-store

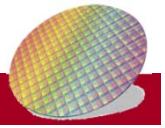
# Types of Instruction Set Architecture



# Code sequences of C=A+B

- Code sequences of C=A+B in four classes of instruction set

| | | | |
|---|---|---|---|
| Push A | Load A | Load R1, A | Load R1, A |
| Push B | Add B | Add R3, R1, B | Load R2, B |
| Add | Store C | Store R3, C | Add R3, R1, R2 |
| Pop C | | | Store R3, C |
| Stack | Accumulator | Register-Memory | Reg-Reg (load-store) |



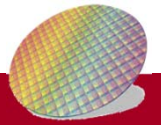
# Why load-store arch. dominates

- Most computer after 1980 uses load-store
- Reason 1: Register is faster than memory
- Reason 2: Register can be used more efficient than memory

(A*B) – (B * C)- (A*D)

May be evaluated by doing the multiplications in any order

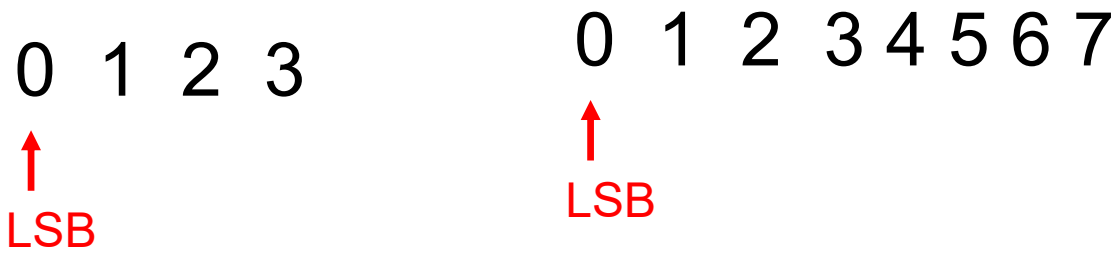
- Reason 3: Memory traffic is reduced when variables are loaded into registers



# Memory Addressing

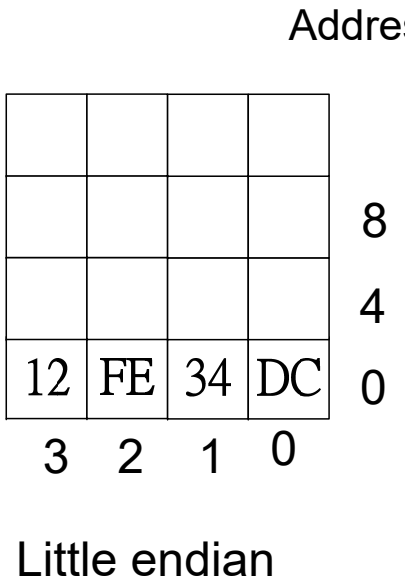
- Two different ways to order bytes: Little and Big Endian
- Little Endian: : least-significant byte at least significant position


- Big Endian : least-significant byte at most significant position

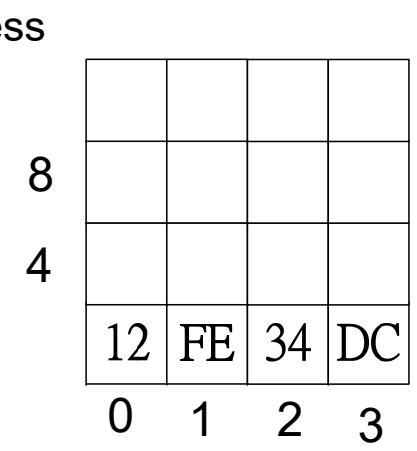


# Example: Big and Little Endian

- How the data is 0x12FE34DC stored in
  1. big endian format
  2. little endian format




Little endian




Big endian

MIPS is Big Endian

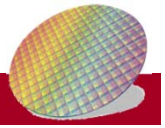
# Addressing Mode

- Addressing mode: how architecture specify the address of an object they will access
- Effective address: when a memory location is used, the actual memory address specified by the addressing mode

add R4, 100 (R1)  Effective Address = 100+ Regs[R1]

- Address modes in MIPS
  - Immediate
  - Register
  - displacement addressing
  - PC-relative addressing
  - (Pseudo) direct addressing

There are other addressing modes, check Figure A.6



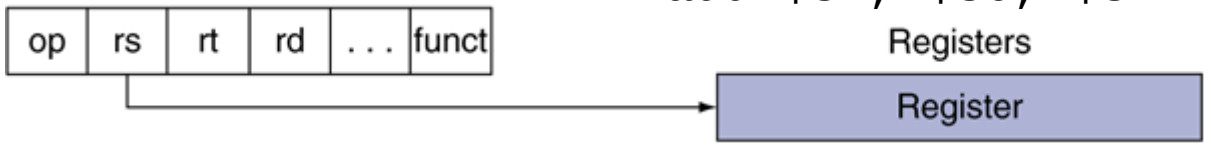
# Addressing Mode

- Immediate addressing: operand is a constant within the instruction (e.g. addi)

```
addi $s1, $s0, 1    # s1 = s0+1
```

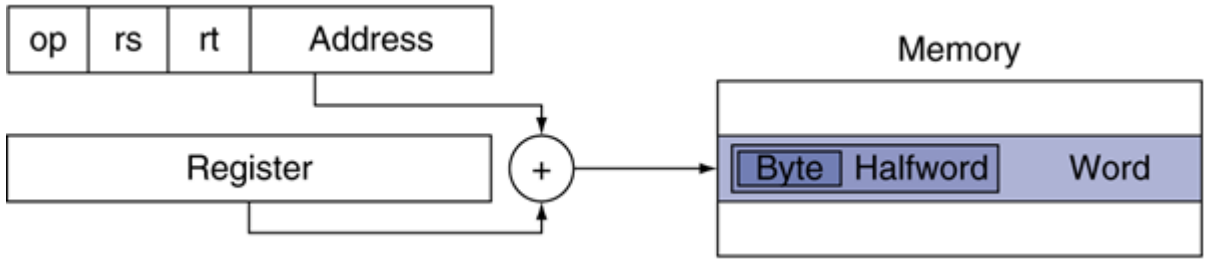

- Register addressing: operand is a register (e.g. add, nor)

```
add $s1, $s0, $s2
```




- Base or displacement addressing: operand is at the memory location (e.g. lw, sw)

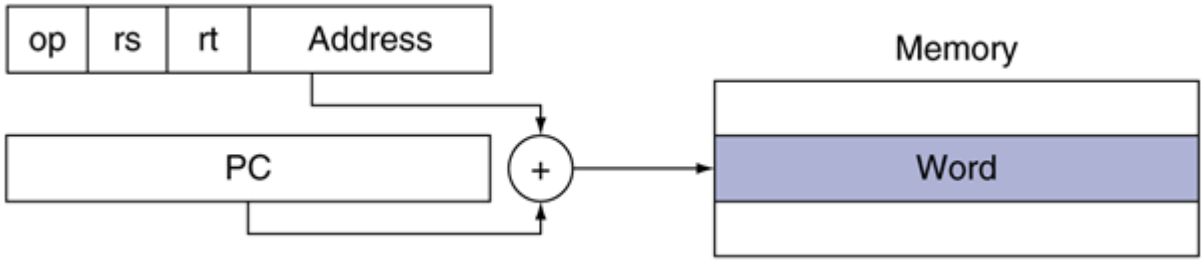
```
lw $t0, 32($s3)
```



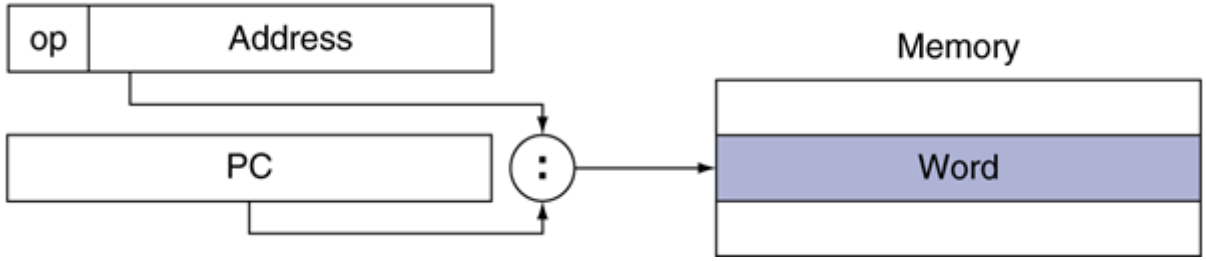
# Addressing Mode (2)

- PC-relative addressing: branch address is the sum of PC and constant (e.g. beq)

beq $s0,$s1, L1




- (Pseudo)direct addressing: jump address is 26 bit of instruction + PC (e.g. j)

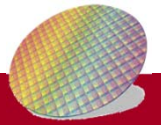


# Type and Size of Operands

- Type of operands normally defined by encoding in opcodes

MIPS machine language

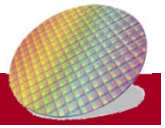
| Name | Format | Example | | | | | | Comments |
|---|---|---|---|---|---|---|---|---|
| add | R | 0 | 18 | 19 | 17 | 0 | 32 | add $s1,$s2 ,$s3 |
| sub | R | 0 | 18 | 19 | 17 | 0 | 34 | sub $s1,$s2 ,$s3 |
| addi | I | 8 | 18 | 17 | 100 | | | addi $s1,$s2 ,100 |
| lw | I | 35 | 18 | 17 | 100 | | | lw $s1,100($s2) |
| sw | I | 43 | 18 | 17 | 100 | | | sw $s1,100($s2) |
| Field size | | 6 bits | 5 bits | 5 bits | 5 bits | 5 bits | 6 bits | All MIPS instructions are 32 bits long |
| R-format | R | op | rs | rt | rd | shamt | funct | Arithmetic instruction format |
| I-format | I | op | rs | rt | address | | | Data transfer format |



# Type and Size of Operands

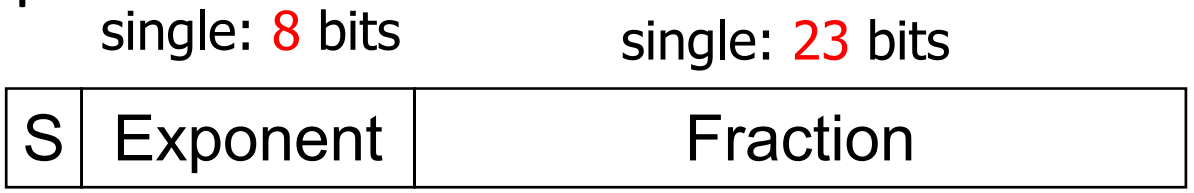
- Common Types:
  - character (8 bits)
  - half word (16 bits)
  - word (32 bits)
  - single-precision floating point (32 bits)
  - double-precision point ( 64 bits)

- Integer is normally represented as two's complement
- Character is ASCII or 16-bit Unicode
- Floating point: IEEE standard 754



# Floating Point Standard- IEEE Std 754-1985

- Single precision - 32-bit

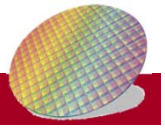



Significand= 1+fraction

$$x = (-1)^S \times (1 + \text{Fraction}) \times 2^{(\text{Exponent} - \text{Bias})}$$

$$x = (-1)^S \times (\text{Significand}) \times 2^{(\text{Exponent} - \text{Bias})}$$

- S: sign bit (0 ⇒ non-negative, 1 ⇒ negative)
- Normalized number $\pm 1.xxxxxxx_2 \times 2^{yyyy}$
  - Always has a leading 1, so no need to represent it explicitly (hidden bit)
- Exponent: excess representation: actual exponent + Bias
  - Ensures exponent is unsigned
  - Single precision: Bias = 127, Double precision: Bias = 1023



# Floating-Point Example – single-precision

What number is represented by the following single-precision float?

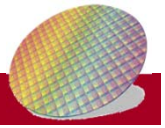
x=1100000010100000000000000$_2$ (32-bit)

- S = 1
- Fraction = 01000…00$_2$
- Exponent = 10000001$_2$ = 129

- x = $(-1)^1 \times (1 + .01_2) \times 2^{(129 - 127)}$

  $= (-1) \times (1+1/4) \times 2^2$

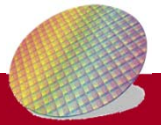
  $= -5.0$



# Instruction Operations

- Operations supported by instructions
- Supported Operations vary widely among architectures.

- Arithmetic and logical:  add, subtract, and , or…
- Data transfer: load-store
- Control: branch, jump..
- System: operating system call, virtual memory management
- Floating point: FP add …
- Decimal: decimal add, decimal multiply (normally for business application)
- String: string move, string compare, string search
- Graphics: pixel and vertex operations…

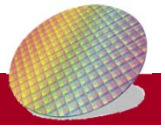


# Encoding an Instruction Set

- Instruction operations and operands are encoded in binary
- The number of registers and the number of addressing modes influence the size of instructions
  - i.e. more registers => more encoded bits for registers
  - i.e. MIPS has 32 registers => 5 bits are needed

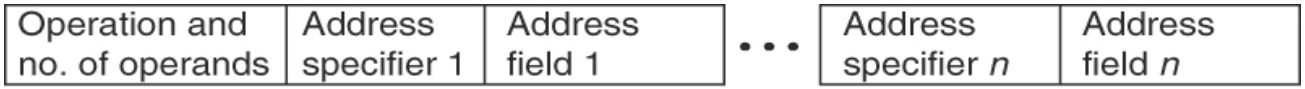
| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Field size | | 6 bits | 5 bits | 5 bits | 5 bits | 5 bits | 6 bits | All MIPS instructions are 32 bits long |
| R-format | R | op | rs | rt | rd | shamt | funct | Arithmetic instruction format |
| I-format | I | op | rs | rt | address | | | Data transfer format |

- Too complicated encoding will affect the performance of the instruction pipeline
  - i.e. ID stage in 5-stage MIPS pipeline will be complicated



# Three basic variation in inst. encoding

- Variable: support any number of operands, address specifier determine the address mode




(a) Variable (e.g., Intel 80x86, VAX)

- Fixed: fixed format, simple but larger code sizes


(b) Fixed (e.g., Alpha, ARM, MIPS, PowerPC, SPARC, SuperH)

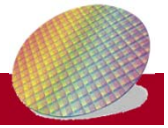
- Hybrid: support multiple format by different inst.




(c) Hybrid (e.g., IBM 360/370, MIPS16, Thumb, TI TMS320C54x)

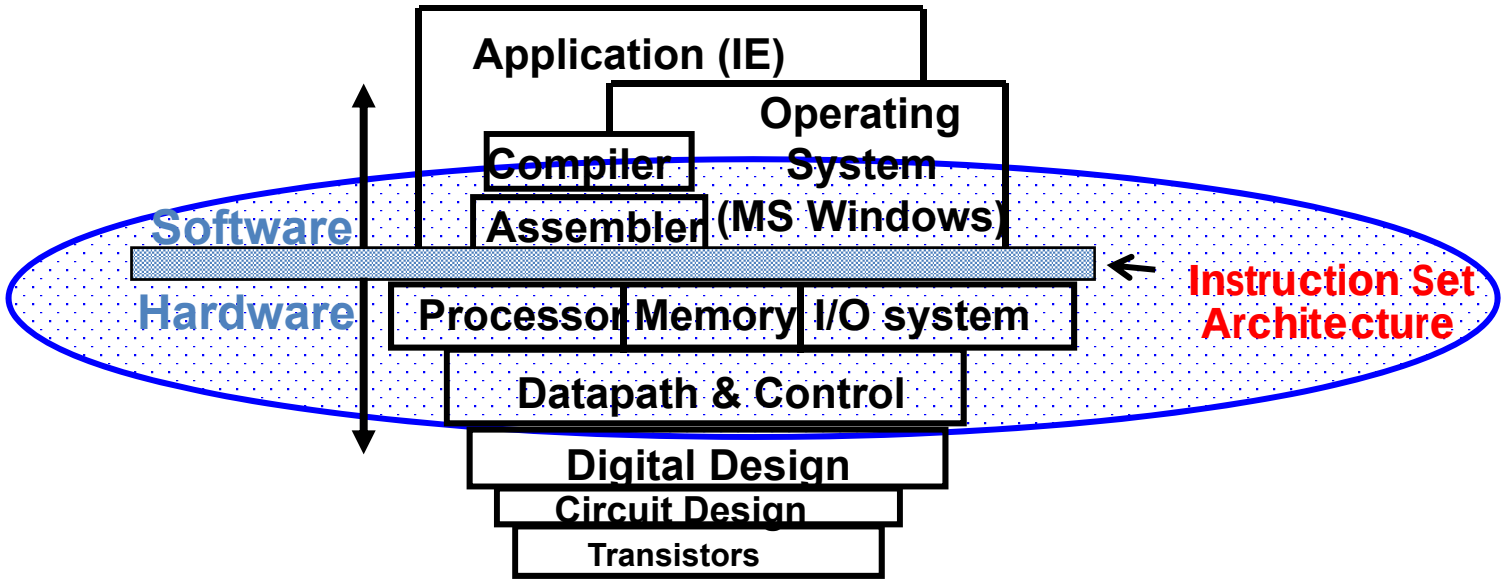
# Quick Summary

- Factors you need to consider when you design an instruction set
  - Addressing Mode
  - Type and size of operands
  - Supported Operation
  - Instruction Encoding

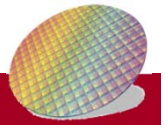


# Instruction Set

- Instruction Set : set of instructions of a computer
- Different computers have different instruction sets
  - But with many aspects in common
- Interface between hardware and software of a computer



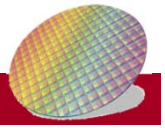
# Review Questions



Three processors P1,P2 and P3 executing the same instruction set.

| | Freq | CPI |
|---|---|---|
| P1 | 3 GHz | 1.5 |
| P2 | 2.5 GHz | 1.0 |
| P3 | 4.0 GHz | 2.2 |

a) Which processor has the highest performance expressed in instructions per second?
b) If the processors each execute a program in 10 seconds, find the number of cycles and the number of instructions.
c) We are trying to reduce the execution time by 30% but this leads to an increase of 20% in the CPI. What clock rate should we have to get this time reduction?



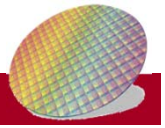
a) Which processor has the highest performance expressed in instructions per second?

a) performance of P1 (instructions/sec) = $3 \times 10^9/1.5 = 2 \times 10^9$
performance of P2 (instructions/sec) = $2.5 \times 10^9/1.0 = 2.5 \times 10^9$
performance of P3 (instructions/sec) = $4 \times 10^9/2.2 = 1.8 \times 10^9$

b. If the processors each execute a program in 10 seconds, find the number of cycles and the number of instructions.

cycles(P1) = $10 \times 3 \times 10^9 = 30 \times 10^9$s
cycles(P2) = $10 \times 2.5 \times 10^9$ s
cycles(P3) = $10 \times 4 \times 10^9 = 40 \times 10^9$ s

No. inst(P1) = $30 \times 10^9 / 1.5 = 20 \times 10^9$
No. inst(P2) = $25 \times 10^9 / 1 = 25 \times 10^9$
No. inst(P3) = $40 \times 10^9/2.2 = 18.18 \times 10^9$



c) We are trying to reduce the execution time by 30% but this leads to an increase of 20% in the CPI. What clock rate should we have to get this time reduction?

$CPI_{new} = CPI_{old} \times 1.2$, then

$CPI(P1) = 1.8$,

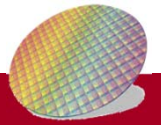
$CPI(P2) = 1.2$, $CPI(P3) = 2.6$

$f$ =No. instr. × CPI/time, then

$f(P1) = 20 \times 10^9 \times 1.8/7 = 5.14\ GH_z$

$f(P2) = 25 \times 10^9 \times 1.2/7 = 4.28\ GH_z$

$f(P1) = 18.18 \times 10^9 \times 2.6/7 = 6.75\ GH_z$



For the following we consider instruction encoding for instruction set architectures.

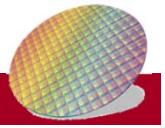
a. Consider the case of a processor with an instruction length of 12 bits and with 32 general-purpose registers so the size of the address fields is 5 bits. Is it possible to have instruction encodings for the following?

- 3 two-address instructions
- 30 one-address instructions
- 45 zero-address instructions

b. Assuming the same instruction length and address field sizes as above, determine if it is possible to have

- 3 two-address instructions
- 31 one-address instructions
- 35 zero-address instructions

Explain your answer.



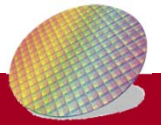
a) There are 32 registers => each address needs 5 bits

| | Addr [11:10] | Addr [9:5] | Addr [4:0] |
|---|---|---|---|
| 3 two-address inst. | 00, 01, 10 | Register | Register |

| | Addr [11:10] | Addr [9:5] | Addr [4:0] |
|---|---|---|---|
| 30 one-address inst. | 11 | 00000~11101 | Register |

| | Addr [11:10] | Addr [9:5] | Addr [4:0] | |
|---|---|---|---|---|
| 45 zero-address inst. | 11 | 11110 | 00000~11111 | 32 inst. |
| | 11 | 11111 | 00000~01100 | 13 inst. |

Therefore, it is possible to have the above instruction encodings



b. There are 32 registers => each address needs 5 bits

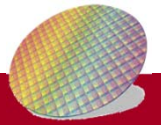
| | Addr [11:10] | Addr [9:5] | Addr [4:0] |
|---|---|---|---|
| 3 two-address inst. | 00, 01, 10 | Register | Register |

| | Addr [11:10] | Addr [9:5] | Addr [4:0] |
|---|---|---|---|
| 31 one-address inst. | 11 | 00000~11110 | Register |

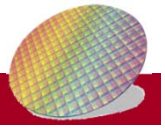
| | Addr [11:10] | Addr [9:5] | Addr [4:0] |
|---|---|---|---|
| 35 zero-address inst. | 11 | 11111 | 00000~11111 |

32 inst.

Can only represent 32 different instruction. Therefore, it is impossible to have these instruction encodings

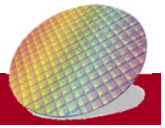


A.10 The design of MIPS provides for 32 general-purpose registers and 32 floating-point registers. If registers are good, are more registers better? List and discuss as many trade-offs as you can that should be considered by instruction set architecture designers examining whether to, and how much to, increase the number of MIPS registers.



Reasons to increase the number of registers include:
1. Greater freedom to employ compilation techniques that consume registers,
such as loop unrolling, common subexpression elimination, and avoiding
name dependences.
2. More locations that can hold values to pass to subroutines.
3. Reduced need to store and re-load values.

Reasons not to increase the number of registers include:
1. More bits needed to represent a register name, thus increasing the overall size
of an instruction or reducing the size of some other field(s) in the instruction.
2. More CPU state to save in the event of an exception.
3. Increased chip area and increased power consumption.



# Backup slides