

CPU design

1. Yield = The proportion of working dies per wafer.
2. Cost per die = $\frac{\text{Cost per wafer}}{\text{Dies per wafer} \times \text{Yield}}$
3. Dies per wafer $\approx \frac{\text{Wafer area}}{\text{Die area}}$ (since wafers are circle)
4. Yield = $\frac{1}{\left[1 + \left(\frac{\text{Defects per area} \times \text{Die area}}{2}\right)\right]^2}$
= Defects per unit area \times Die area

Power = Capacitive load \times Voltage² \times Frequency

ISA

Components: processor, I/O, mem, and network.

ISA: formal specification of the instruction set that is implemented in the machine hardware.

1. Simplicity favours regularity; 2. Smaller is faster; 3. Make the common case fast; 4. Good design demands good compromises;

Important registers:

(ra: usually PC + 4); (sp: must be aligned to 4 bytes);
(gp: holds the base address of global variables)

Arithmetic

rs1 and rs2 fields kept in the same place: imm field in S-type separated

destination = source1 op source2

In I format, values range: -2^{11} to $2^{11} - 1$.

Load 32 bits:

lui t0, 1010 1010 1010 1010 1010b

ori t0, t0, 1010 1010 1010b

logical shift: fill the vacancy with zeros

slli t2, s0, 8 # t2 = s0 << 8 bits

srli t2, s0, 8 # t2 = s0 >> 8 bits

lw t0, 4(s3) # load word from mem to reg

sw t0, 8(s3) # store word from reg to mem

(loaded or stored using a 5-bit address)

NOTE: Address is byte-base: increment 4 when accessing reg

Little Endian: rightmost byte is the most significant byte.

1b places the byte from mem into the rightmost 8 bits of the dest reg and signed extension.

1b t0, 1(s3) # load byte from memory

sb t0, 6(s3) # store byte to memory

stack grows from high address to low address

2's complement: complement all the bits and then add 1

6 = 00... 0110₂ \Rightarrow 11... 1001₂ + 1 \Rightarrow 11... 1010 = -6

n-bit signed binary: $[2^{n-1} - 1, -2^{n-1}]$

ALU

32-bit signed numbers: range from $2^{31} - 1$ to -2^{31}

If the bit string represents address: 0 to $2^{32} - 1$.

Sign extension copies the most significant bit into the other bits to preserve the sign of the number.

Ripple Carry Adder: connect all adders in sequence, slow because each bit's carry-out depends on the previous bit's carry-in, leading to a cumulative delay.

Overflow: adding two positive numbers yields a negative / adding two negative numbers gives a positive / subtracting a negative from a positive gives a negative / subtracting a positive from a negative gives a positive.

mul: 32-bit \times 32-bit multiplication and places the lower 32 bits in the destination register. mulh, mulhu, and mulhsu perform the same multiplication but return the upper 32 bits of the full 64-bit product.

Logical shifts fill with zeros, while arithmetic right shifts fill with the sign bit.

Floating $\underbrace{6.6254}_{\text{Mantissa (always +)}} \times \underbrace{10}_{\text{Base}}^{-27} \Leftarrow \pm 1.M \times 2^{E'-127}$

Structure: S — E' — M

S: Sign bit; E': 8-bit signed exponent; M: mantissa

e.g. 40C0000₁₆ in decimal

1. 40C0000 = 0 10000001 100000000000000000000000

2. Sign bit (0): Positive (+)

3. Exponent: 10000001₂ - 127 = 129 - 127 = 2

4. Mantissa: 1.1000000000... = 1 + 1 \times 2⁻¹ = 1.5

Result: 1.5 \times 2² = 6

e.g. -0.5₁₀ in binary

1. Sign bit: 1

2. Mantissa: 0.5 = 1.0 \times 2⁻¹

3. Exponent: 127 - 1 = 126 = 01111110

Result: -0.5₁₀ = 10111111000000000000000000000000

$E = 0, M = 0$: 0; $E = 0, M \neq 0$: denormalized number, which is $\pm 0.M \times 2^{-126}$; $E = 1 \dots 1, M = 0$: $\pm \infty$, depending on the sign; $E = 1 \dots 1, M \neq 0$: NaN (Not a Number).

Datapath

combinational: ALU; state: memory

The partition of imm field: align the imm with other instruction types - more efficient implementation of control units.

The instruction is decoded in the path between the Instruction Memory and Register File.

Pipeline

clock cycle: timed to accommodate the slowest instruction: instr take same amount of time

CPU time = CPI \times CC \times IC, CPI = cycles per instruction, CC = clock cycle time, IC = instruction count.

IF: Instruction fetch and PC update

ID: Instruction decode and register file read

EXE: Execution or address calculation

MEM: Data memory access (only in loads and stores)

WB: Write the result data back into the register file
instruction latency (time from start of instr to completion) is not reduced.

State registers - between pipeline stage. Register: flip-flop, data moves in at rising edge

Structural hazards: conflicts in the use of a resource

- separating instruction and data memories

- reads in the 2nd half of the cycle and writes in the 1st half