CPT_S 260 Intro to Computer Architecture
Lecture 42

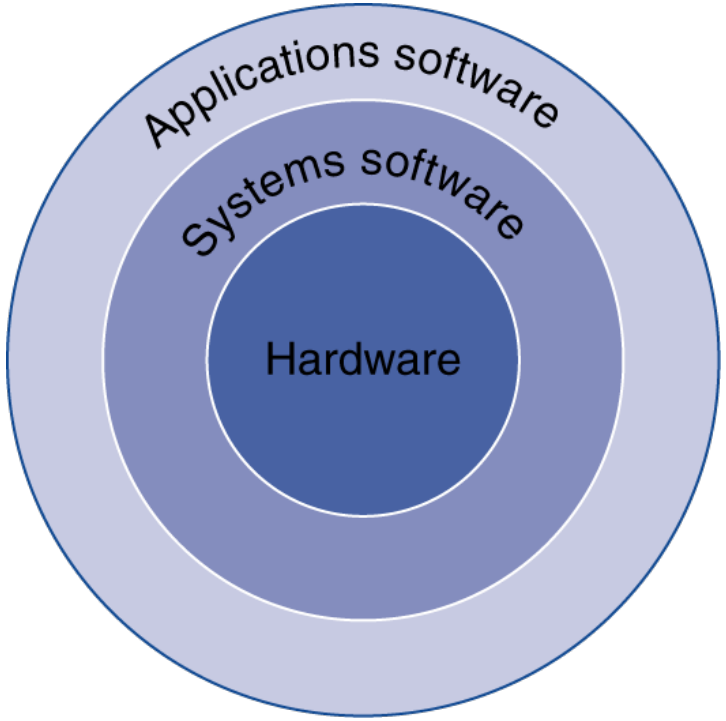
Final Review

Ganapati Bhat
School of Electrical Engineering and Computer Science
Washington State University

Eight Great Ideas in Computer Architecture

- Design for *Moore's Law*
- Use *abstraction* to simplify design
- Make the *common case fast*
- Performance *via parallelism*
- Performance *via pipelining*
- Performance *via prediction*
- *Hierarchy* of memories
- *Dependability via* redundancy

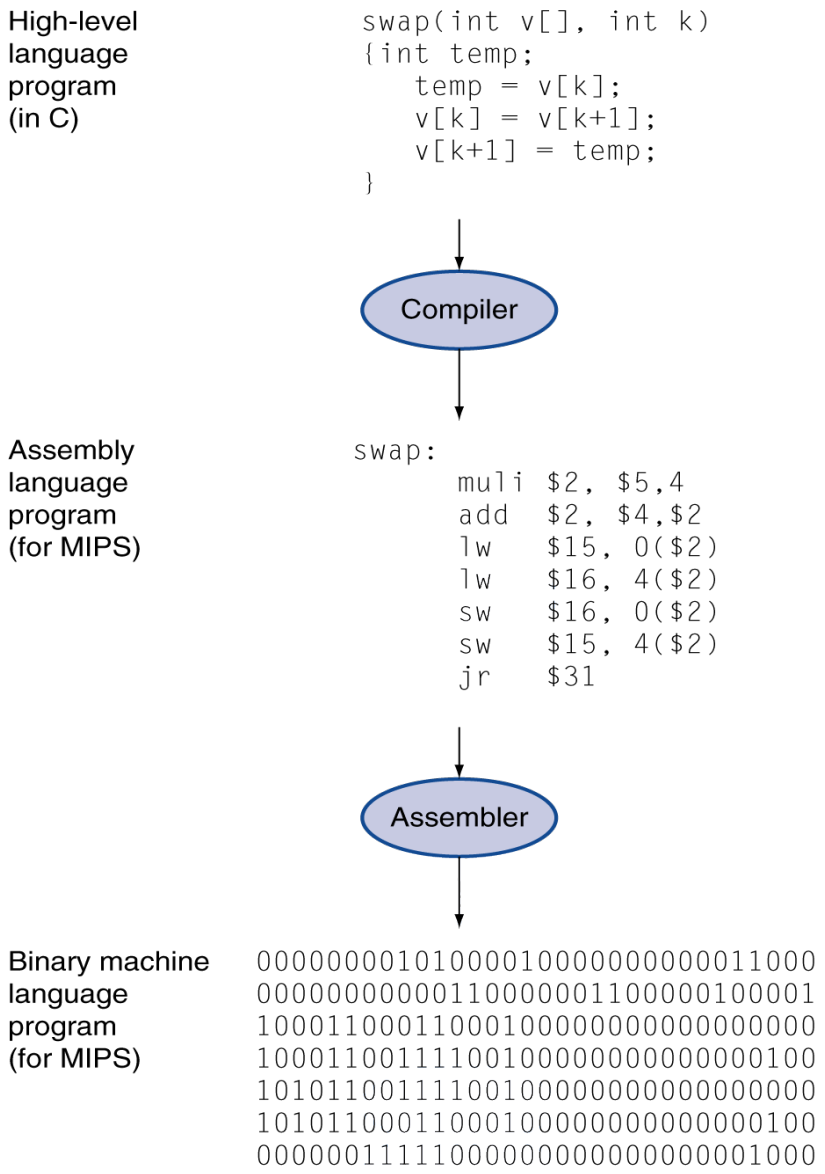
Below Your Program



- **Application software**
  - Written in high-level language
- **System software**
  - Compiler: translates HLL code to machine code
  - Operating System: service code
    - » Handling input/output
    - » Managing memory and storage
    - » Scheduling tasks & sharing resources
- **Hardware**
  - Processor, memory, I/O controllers

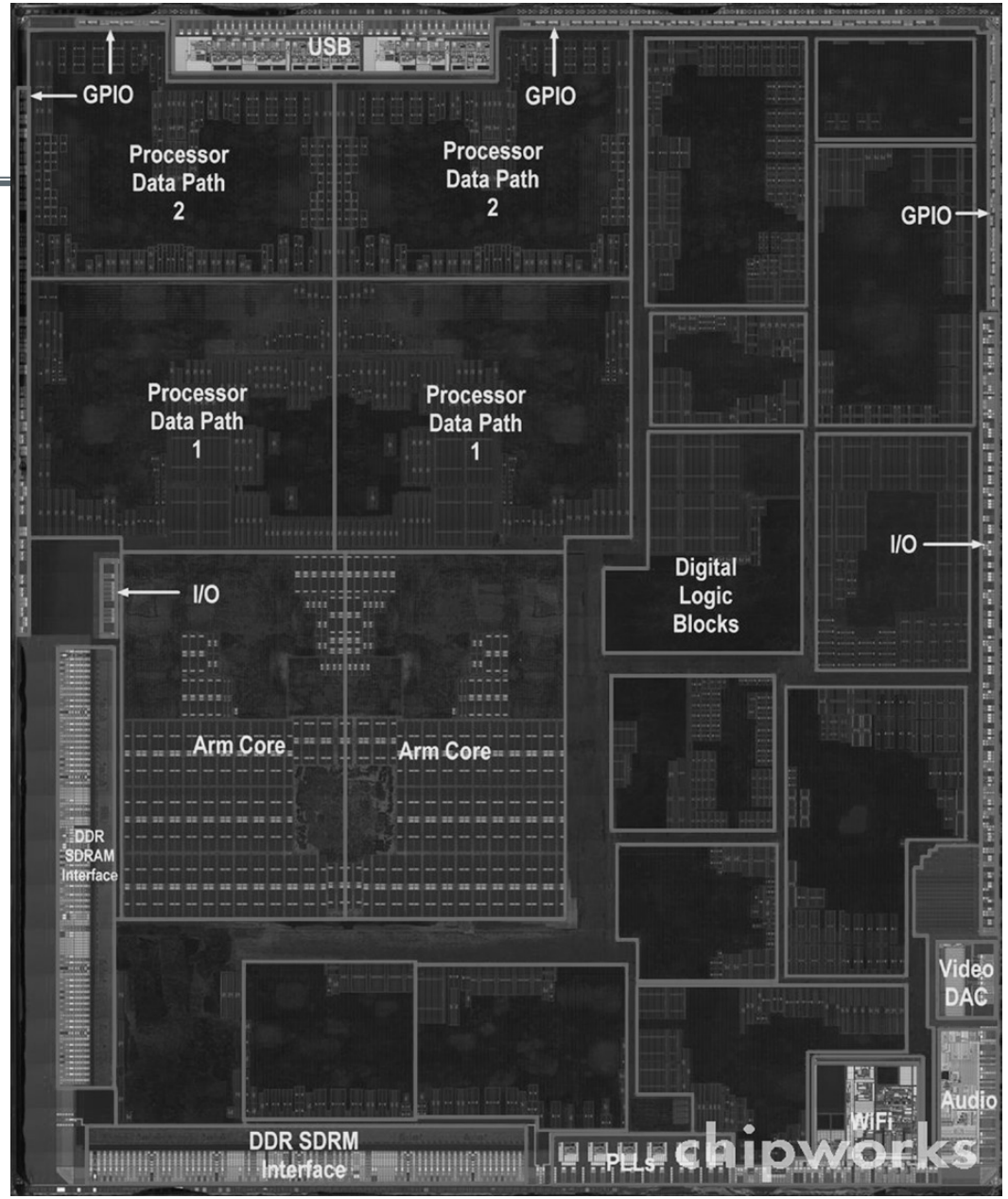
Levels of Program Code

- **High-level language**
  - Level of abstraction closer to problem domain
  - Provides for productivity and portability
- **Assembly language**
  - Textual representation of instructions
- **Hardware representation**
  - Binary digits (bits)
  - Encoded instructions and data

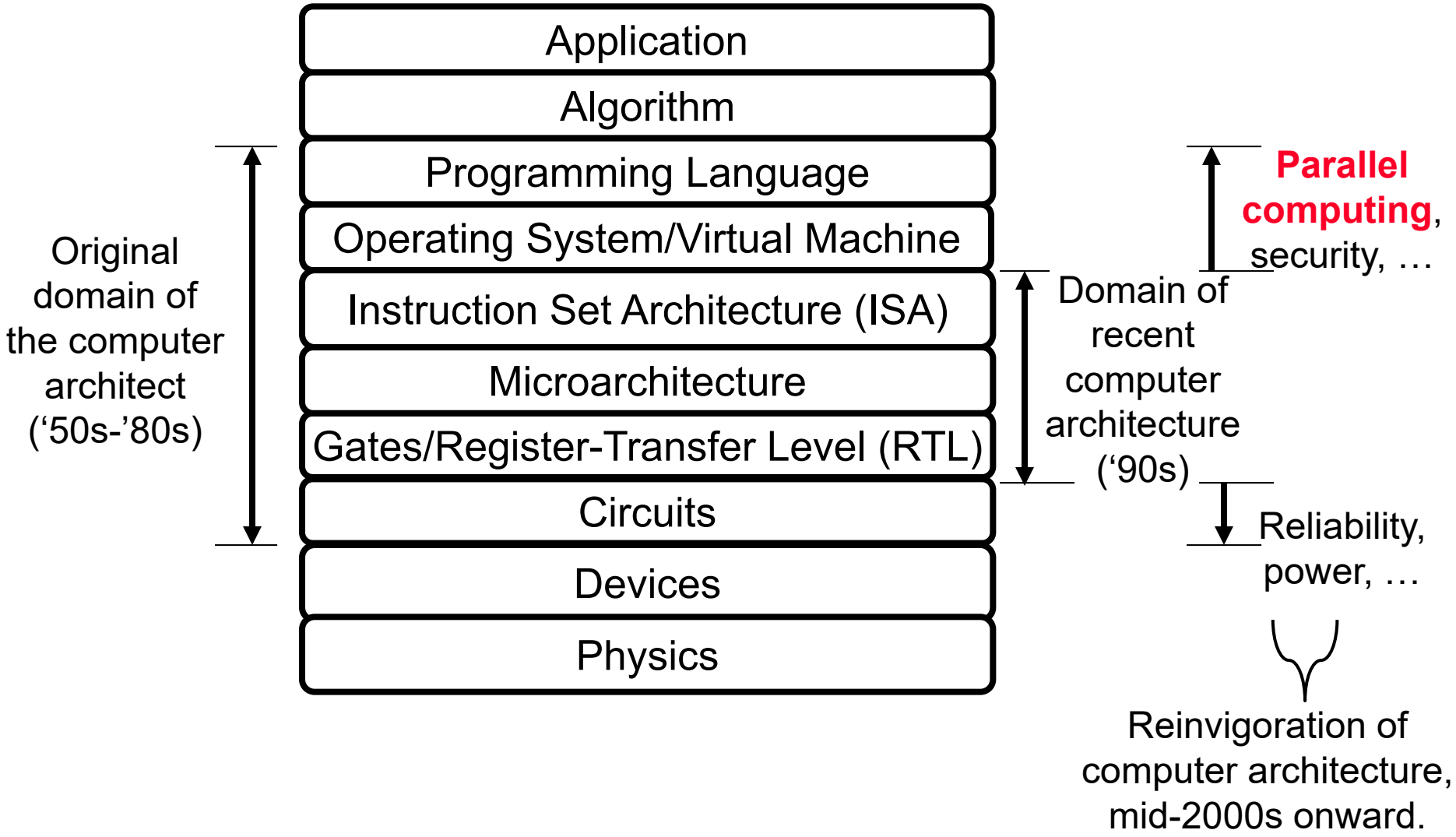


Inside the Processor

- Apple A5



Abstraction Layers in Modern Systems



Performance Summary

$$\text{CPU Time} = \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Clock cycles}}{\text{Instruction}} \times \frac{\text{Seconds}}{\text{Clock cycle}}$$

- **Performance depends on**
  - Algorithm: affects IC, possibly CPI
  - Programming language: affects IC, CPI
  - Compiler: affects IC, CPI
  - Instruction set architecture: affects IC, CPI, T_c

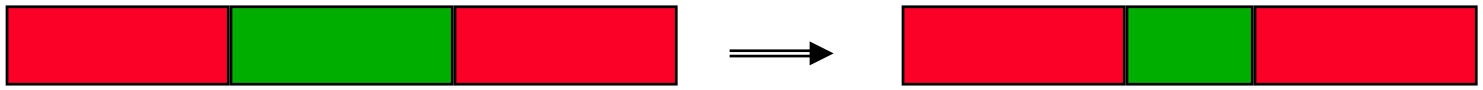
Amdahl's Law

- **How do we increase performance?**
  - Utilize parallelism
  - Principle of locality
  - Focus on the common case
- **Amdahl's law provides a method to quantify speedup**

$$Speedup_{overall} = \frac{t_{old}}{t_{new}} = \frac{1}{(1 - fraction_{enhanced}) + \frac{fraction_{enhanced}}{speedup_{enhanced}}}$$

- **Best achievable speedup is**

$$Speedup_{maximum} = \frac{1}{1 - fraction_{enhanced}}$$



Power and Energy

- **Energy to complete operation (Joules)**
  - Corresponds approximately to battery life
  - (Battery energy capacity actually depends on rate of discharge)
- **Peak power dissipation (Watts = Joules/second)**
  - Affects packaging (power and ground pins, thermal design)
  - Thermal considerations determine the peak power -> TDP
- **di/dt, peak change in supply current (Amps/second)**
  - Affects power supply noise (power and ground pins, decoupling capacitors)
- **Components of power consumption**

$$P = P_{dyn} + P_{static}$$
$$= \alpha C V^2 f + I_{leak} V$$

Numbering Systems

- **A number system of a specific base (radix) uses numbers from 0 to that base-1**
- **Numbers can be computed to decimal through the sum of the weighted digits-**

$$\boldsymbol{Number} = \sum_{i=0}^{n} base^{i} * digit$$

	Binary	Octal	Decimal	Hexadecimal
Base	2	8	10	16
Symbols	{0,1}	{0,1,2,3,4,5,6,7}	{0,1,2,3,4,5,6,7,8,9}	{0,1,2,3,4,5,6,7,8,9, A,B,C,D,E,F}

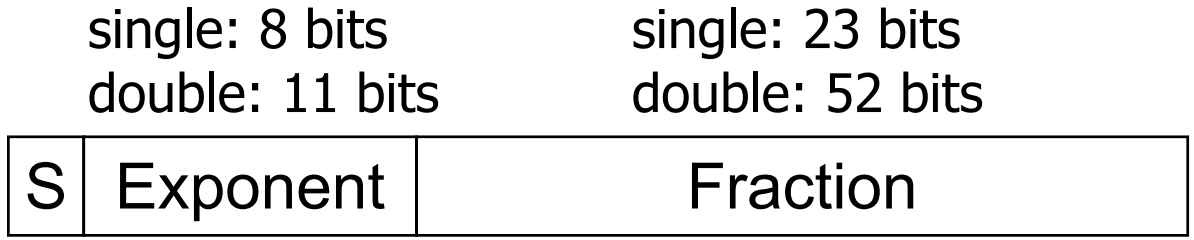
2's-Complement Signed Integers

- **Given an n-bit number:**

$$x = -x_{n-1}2^{n-1} + x_{n-2}2^{n-2} + \cdots + x_1 2^1 + x_0 2^0$$

- **Range: -2^{n-1} to $+2^{n-1} - 1$**
- **Example**
  - $1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1100_2$
    $= -1\times2^{31} + 1\times2^{30} + \ldots + 1\times2^2 + 0\times2^1 + 0\times2^0$
    $= -2{,}147{,}483{,}648 + 2{,}147{,}483{,}644 = -4_{10}$
- **Using 32 bits, 2's complement range is:**
  - –2,147,483,648 to +2,147,483,647

IEEE Floating-Point Format



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

- **S: sign bit (0 ⇒ non-negative, 1 ⇒ negative)**
- **Normalize significand: 1.0 ≤ |significand| < 2.0**
  - Always has a leading pre-binary-point 1 bit, so no need to represent it explicitly (hidden bit)
  - Significand is Fraction with the "1." restored
- **Exponent: excess representation: actual exponent + Bias**
  - Ensures exponent is unsigned
  - Single: Bias = 127; Double: Bias = 1023

Arithmetic Example

- C code:

```
f = (g + h) - (i + j);
```

- Compiled MIPS code:

```
add t0, g, h   # temp t0 = g + h
add t1, i, j   # temp t1 = i + j
sub f, t0, t1  # f = t0 - t1
```

- We see what are these variables!

Register Operands

- **Arithmetic instructions use register operands**
- **MIPS has a 32 × 32-bit register file**
  - Use for frequently accessed data
  - Numbered 0 to 31
  - 32-bit data called a “word”
- **Assembler names**
  - $t0, $t1, …, $t9 for temporary values
  - $s0, $s1, …, $s7 for saved variables
- ***Design Principle 2:* Smaller is faster**
  - c.f. main memory: millions of locations

Memory Operand Example #1

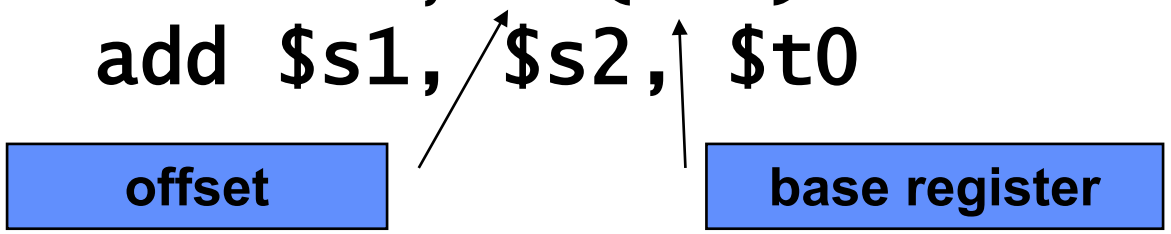
- **C code:**

```
g = h + A[8];
```

  - Assume that g is in $s1, h in $s2, base address of A in $s3

- **Compiled MIPS code:**
  - Index 8 requires offset of 32
    - » 4 bytes per word

```
lw  $t0, 32($s3)    # load word
add $s1, $s2, $t0
```



Memory Operand Example #2

- **C code:**

```
A[12] = h + A[8];
```

  - h in $s2, base address of A in $s3

- **Compiled MIPS code:**
  - Index 8 requires offset of 32

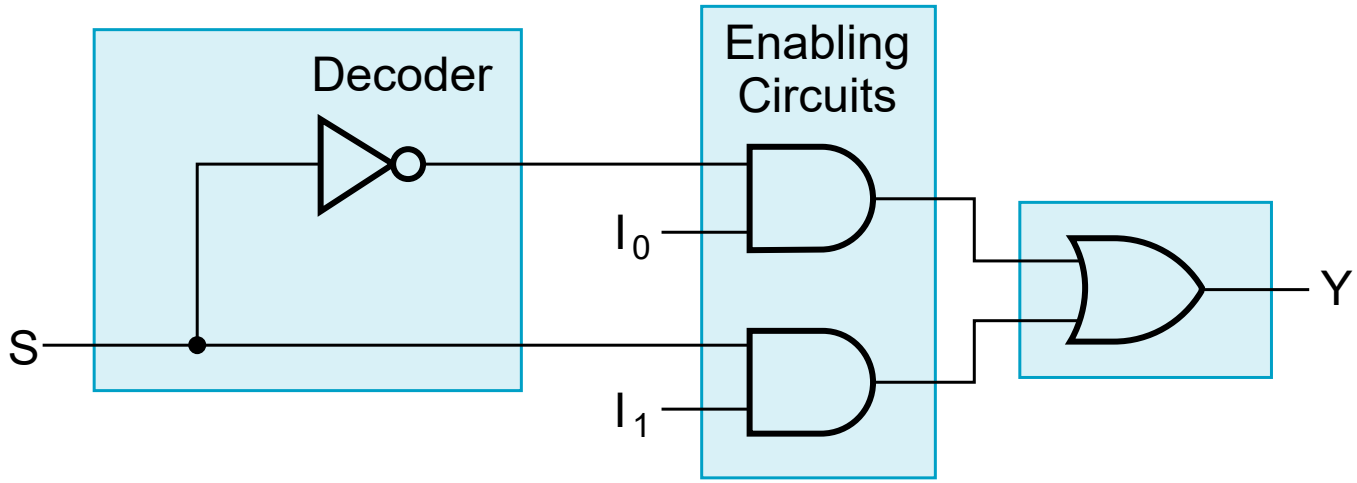
```
lw  $t0, 32($s3)    # load word
add $t0, $s2, $t0
sw  $t0, 48($s3)    # store word
```

2-to-1-Line Multiplexer

- **Since 2 = 2^1, n = 1**
- **The single selection variable S has two values:**
  - S = 0 selects input I_0
  - S = 1 selects input I_1
- **The equation:**

$$Y = \bar{S}I_0 + SI_1$$

- **The circuit:**



Rules of Boolean Algebra

- **Associative Law of multiplication**

$$A \cdot (B \cdot C) = (A \cdot B) \cdot C$$

- **Distributive Law of multiplication**

$$A + BC = (A + B) \cdot (A + C)$$

- **Annulment law:**

$$A \cdot 0 = 0$$
$$A + 1 = 1$$

- **Identity law:**

$$A \cdot 1 = A$$
$$A + 0 = A$$

Rules of Boolean Algebra

- **Complement law:**

$$A + \bar{A} = 1$$
$$A \cdot \bar{A} = 0$$

- **Double negation law:**

$$\bar{\bar{A}} = A$$

- **Absorption law:**

$$A \cdot (A + B) = A$$
$$A + AB = A$$
$$A + \bar{A}B = A + B$$

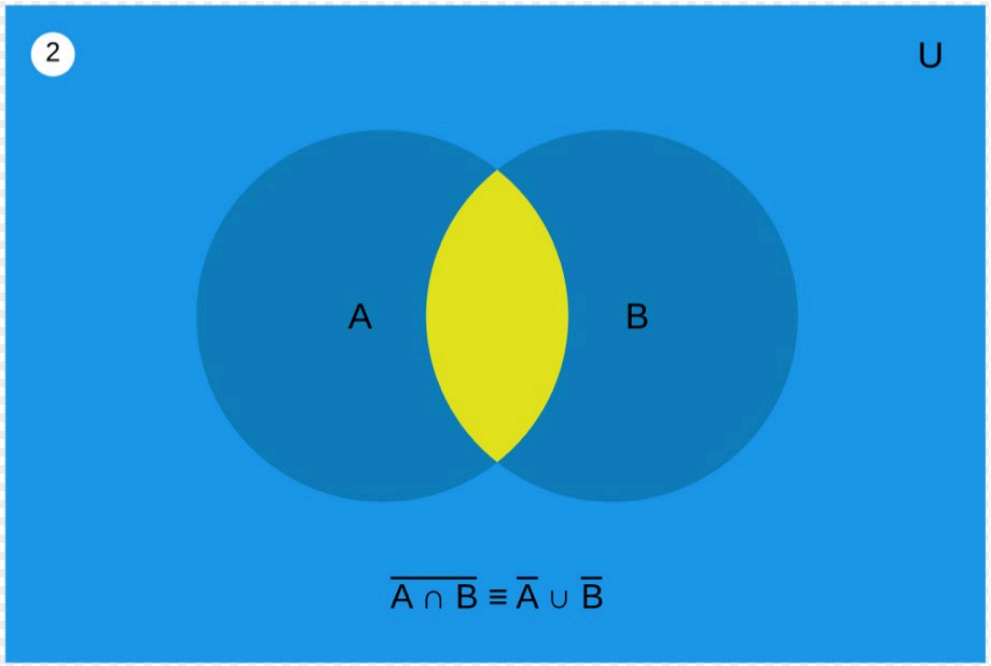
- **Idempotent law:**

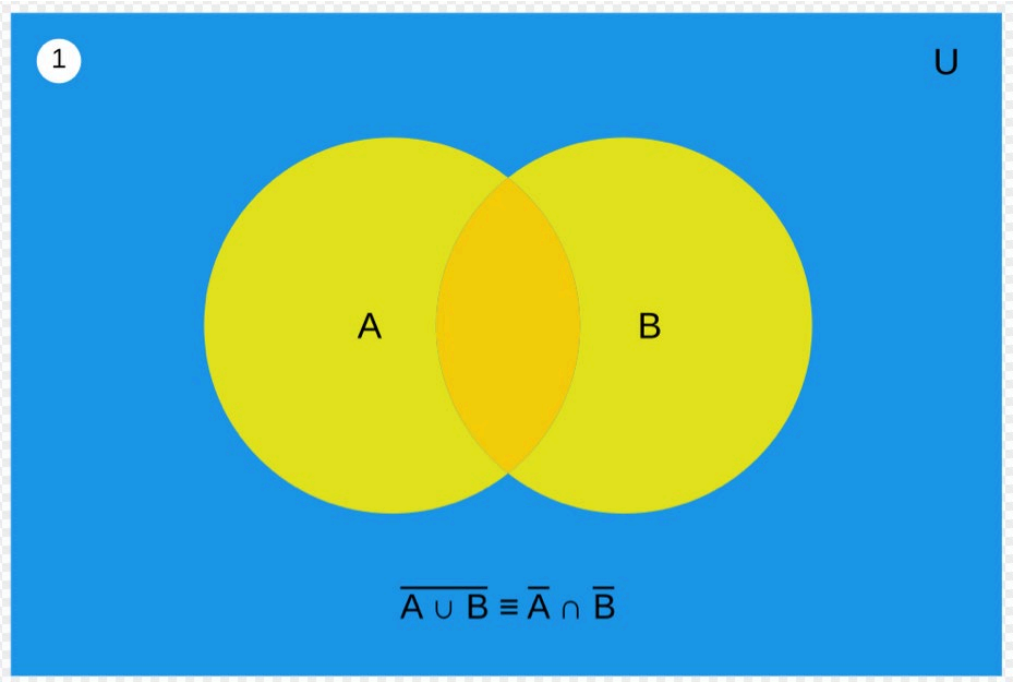
$$A + A = A$$
$$A \cdot A = A$$

De Morgan's Laws

- **Transformation rules that help simplification of negations**
- **Statement:**

$$\overline{AB} = \bar{A} + \bar{B}$$
$$\overline{(A + B)} = \bar{A} \cdot \bar{B}$$





Sum of Products

- **Minterm Expressions**
- **If input is 0 we take the complement of the variable**
- **If input is 1 we take the variable as is**
- **To get the desired canonical SOP expression we will add the minterms (product terms) for which the output is 1**

$$F = \bar{A}B + A\bar{B} + AB$$

A	B	F	Minterm
0	0	0	A'B'
0	1	1	A'B
1	0	1	AB'
1	1	1	AB

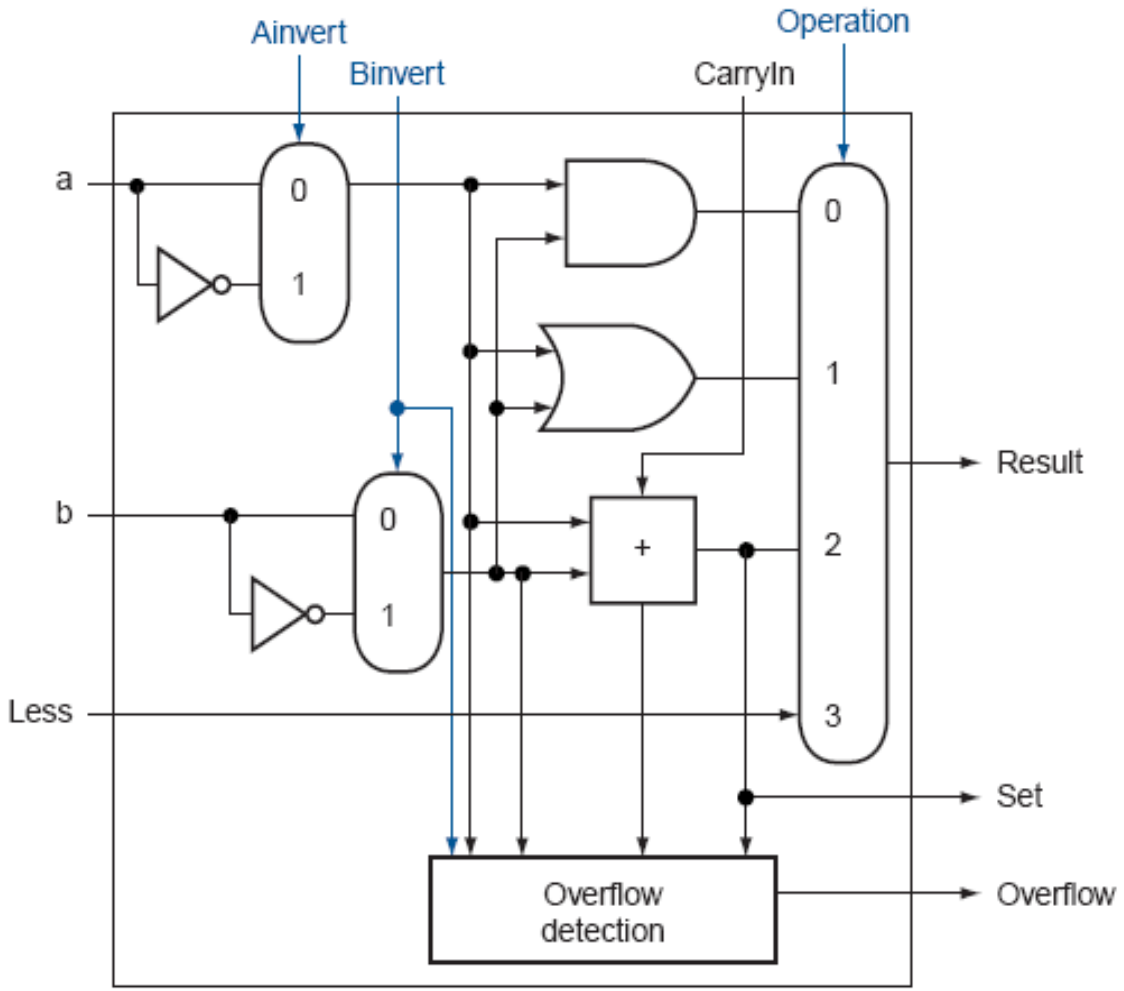
Product of Sums

- **Maxterm Expressions**
- **If input is 1, we take the complement of the variable**
- **If input is 0, we take the variable as is**
- **To get the desired canonical POS expression we will multiply the maxterms (sum terms) for which the output is 0**

$$F = (A + B) . (\bar{A} + \bar{B})$$

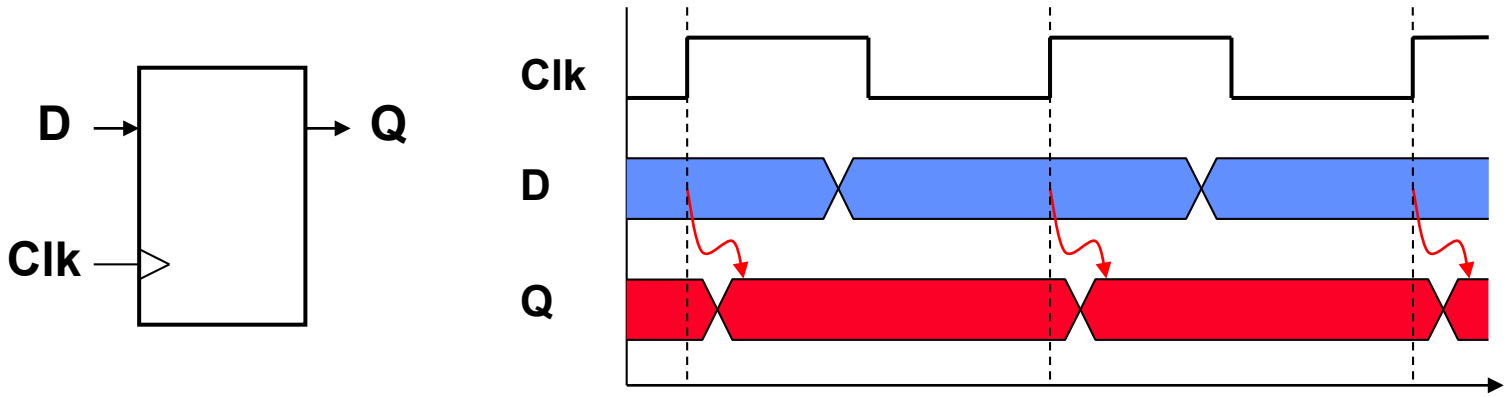
A	B	F	Minterm
0	0	0	A'B'
0	1	1	A'B
1	0	1	AB'
1	1	1	AB

1-bit ALU


Ainvert
Binvert
CarryIn
Operation
a
b
Less
0
1
0
1
0
1
2
3
+
Result
Set
Overflow
Overflow detection

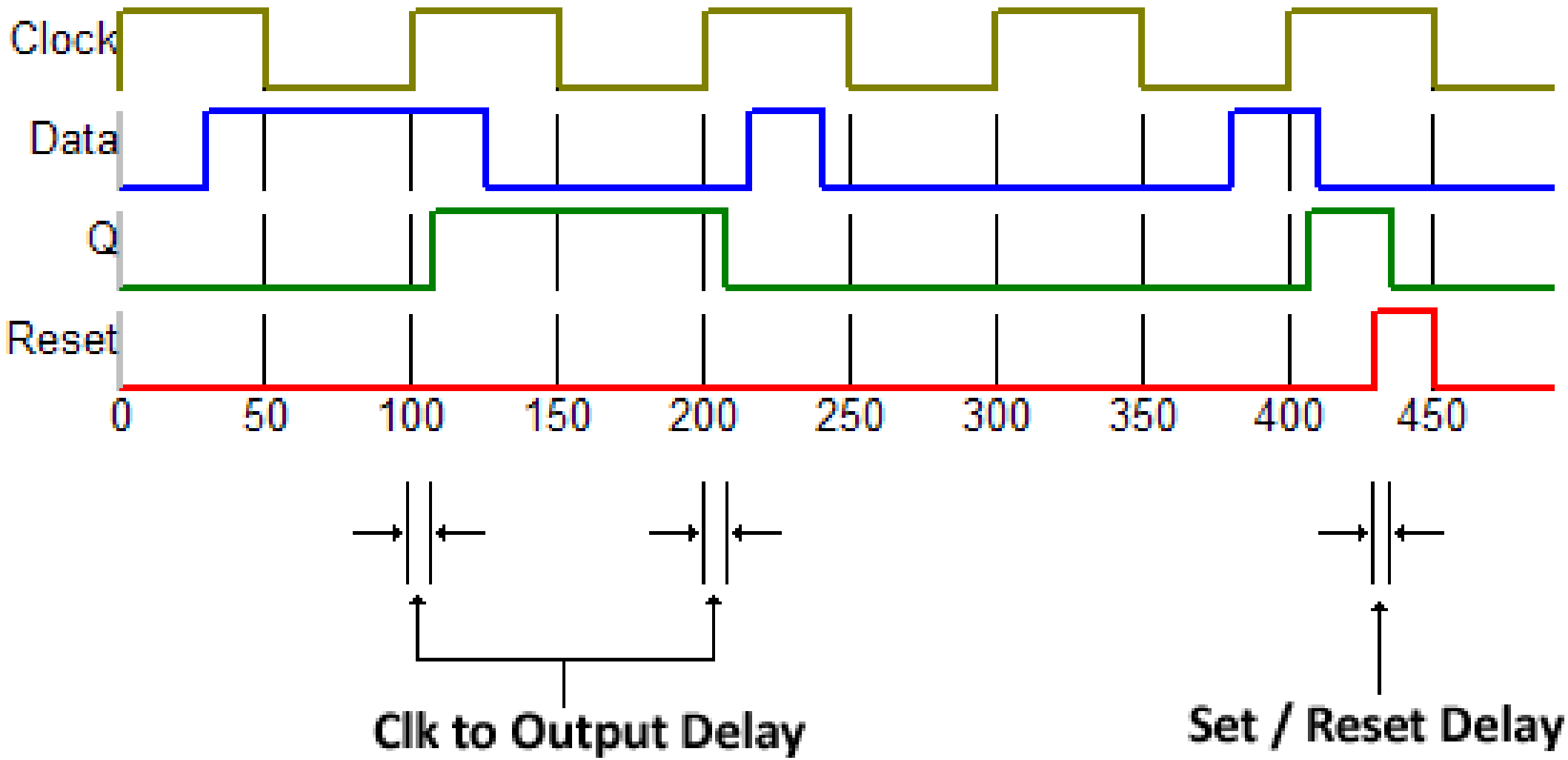
Sequential Elements

- **Flip flops: stores data in a circuit**
  - Uses a clock signal to determine when to update the stored value
  - Edge-triggered: update when Clk changes from 0 to 1



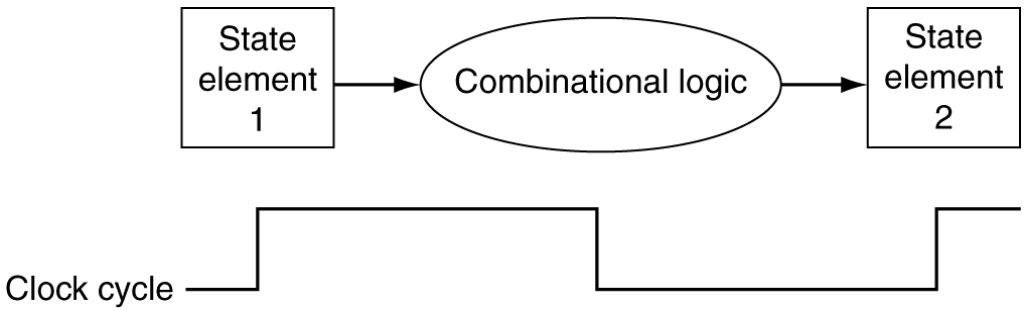
24

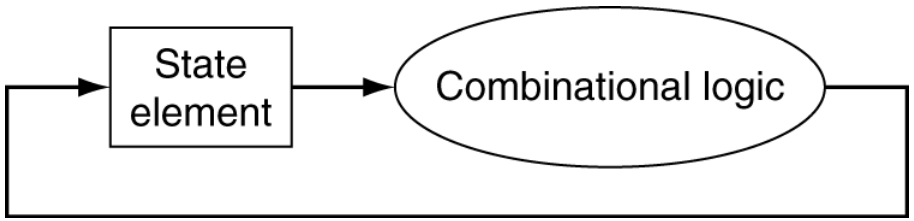
Clocking Sequence



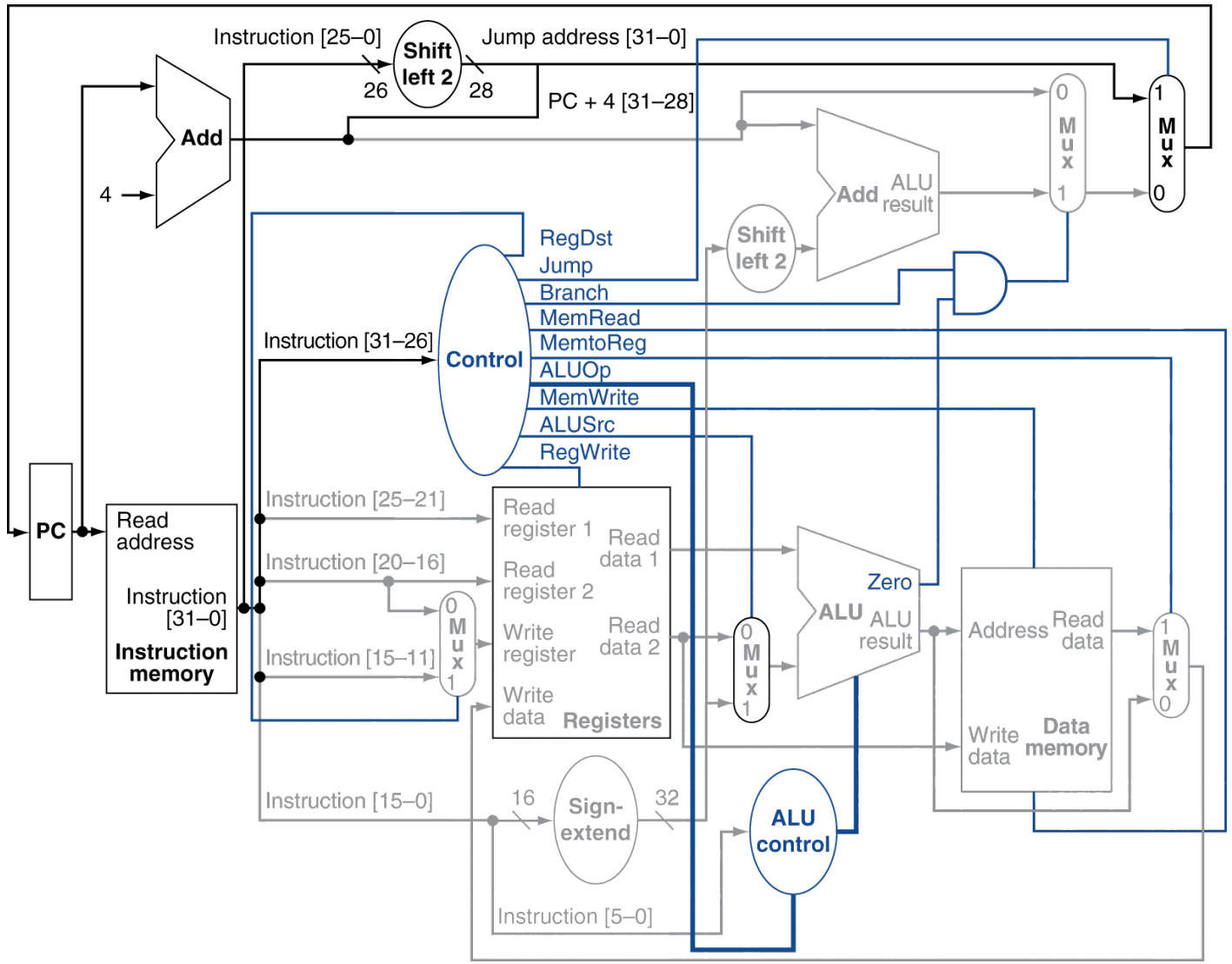
Clocking Methodology

- **Combinational logic transforms data during clock cycles**
  - Between clock edges
  - Input from state elements, output to state element
  - Longest delay determines clock period

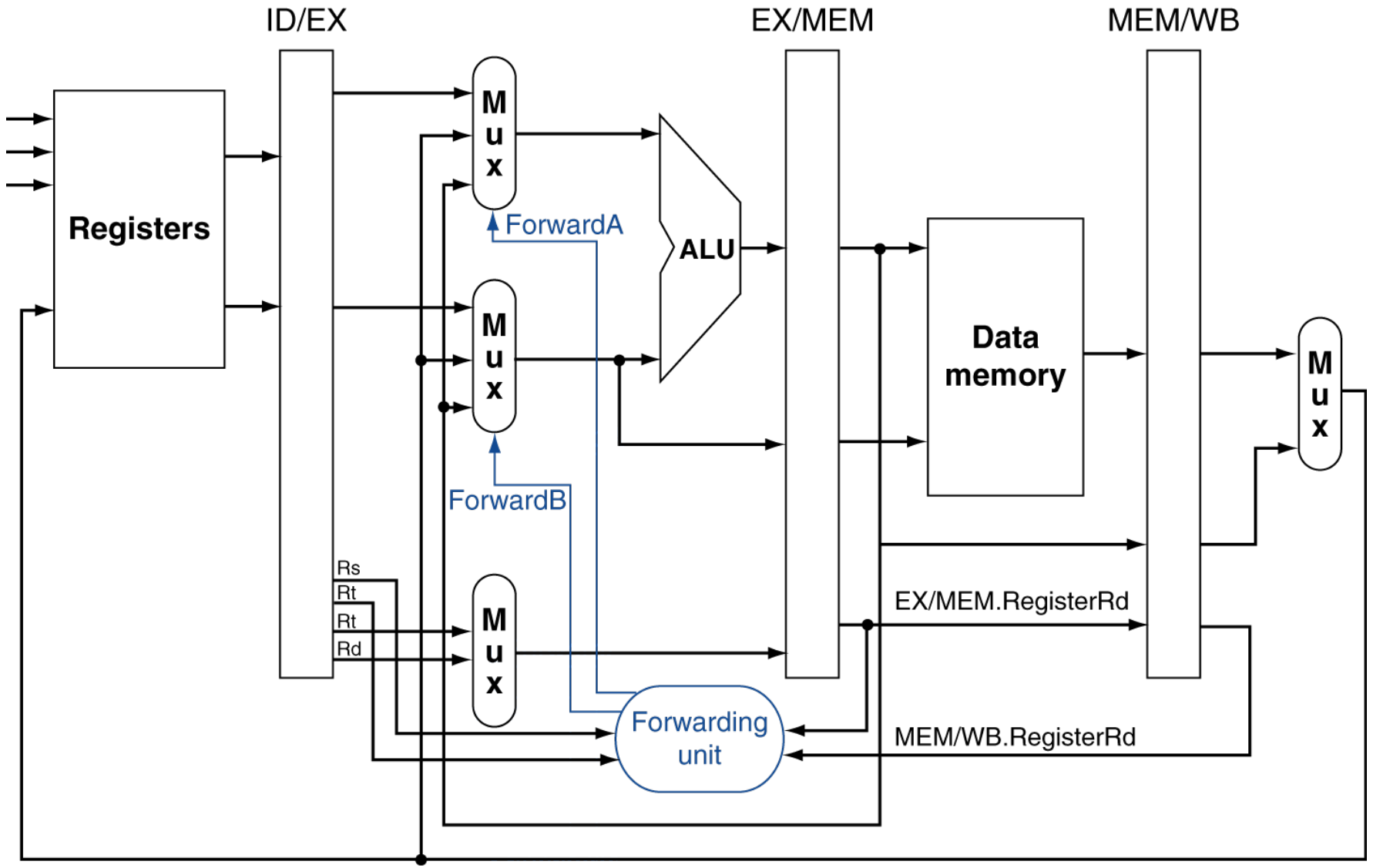




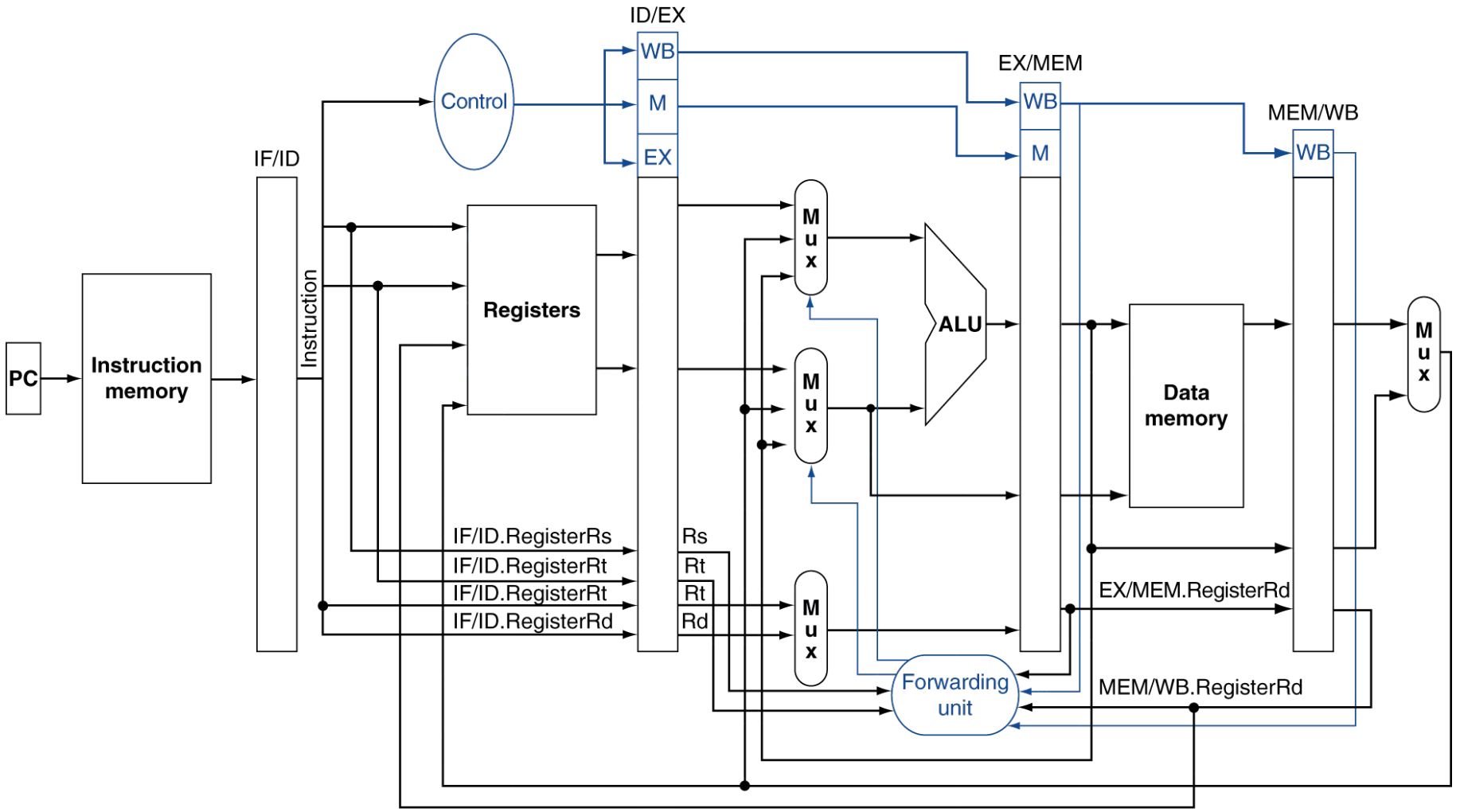
MIPS Datapath



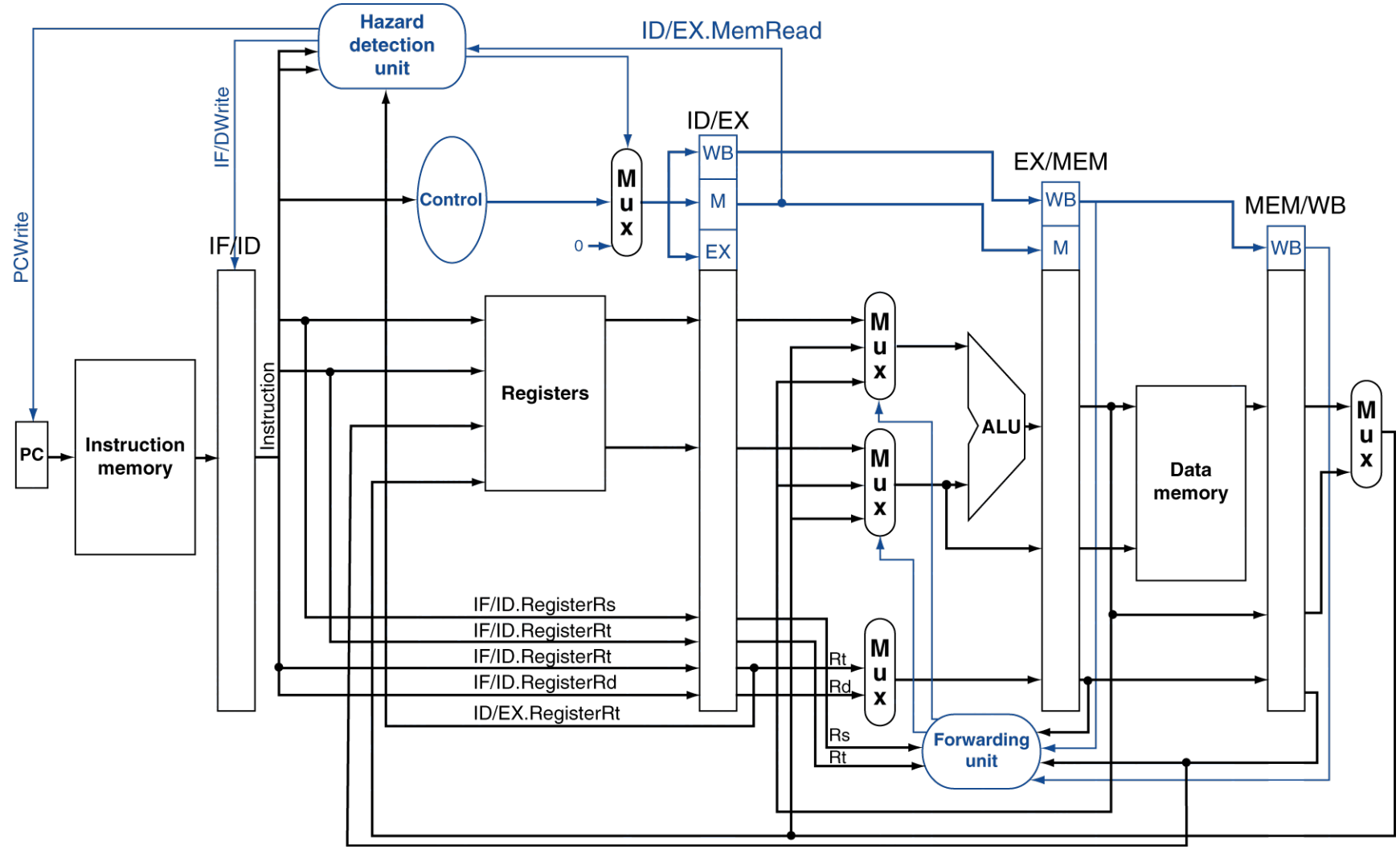
Forwarding Paths



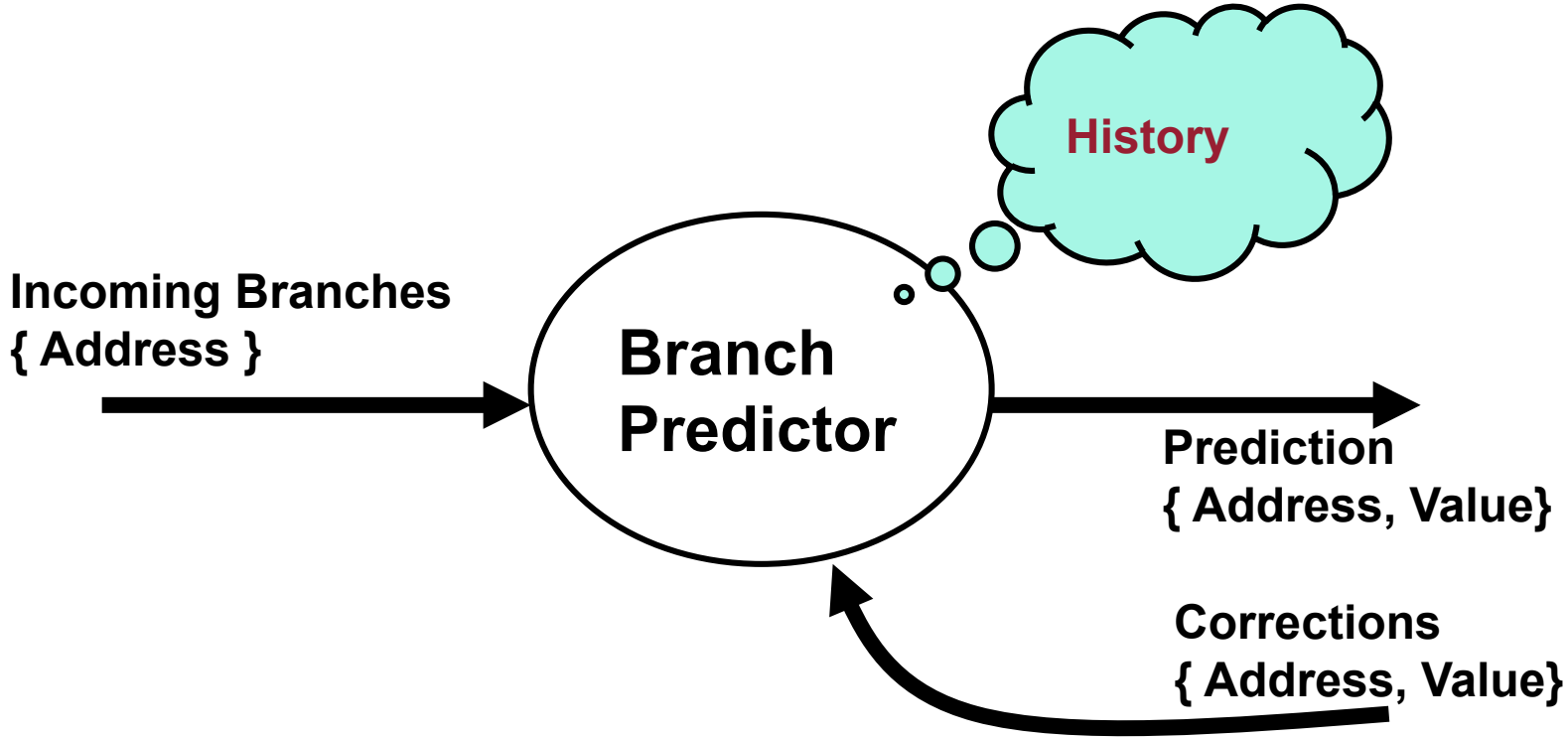
Datapath with Forwarding



Datapath with Hazard Detection



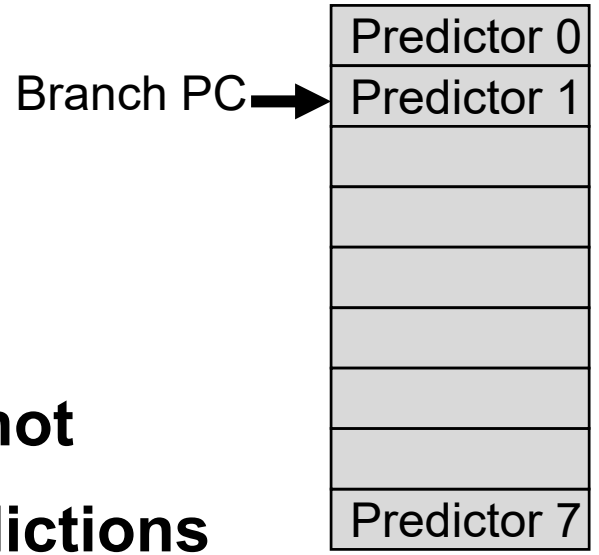
Dynamic Branch Prediction Problem



- Incoming stream of addresses
- Fast outgoing stream of predictions
- Correction information returned from pipeline

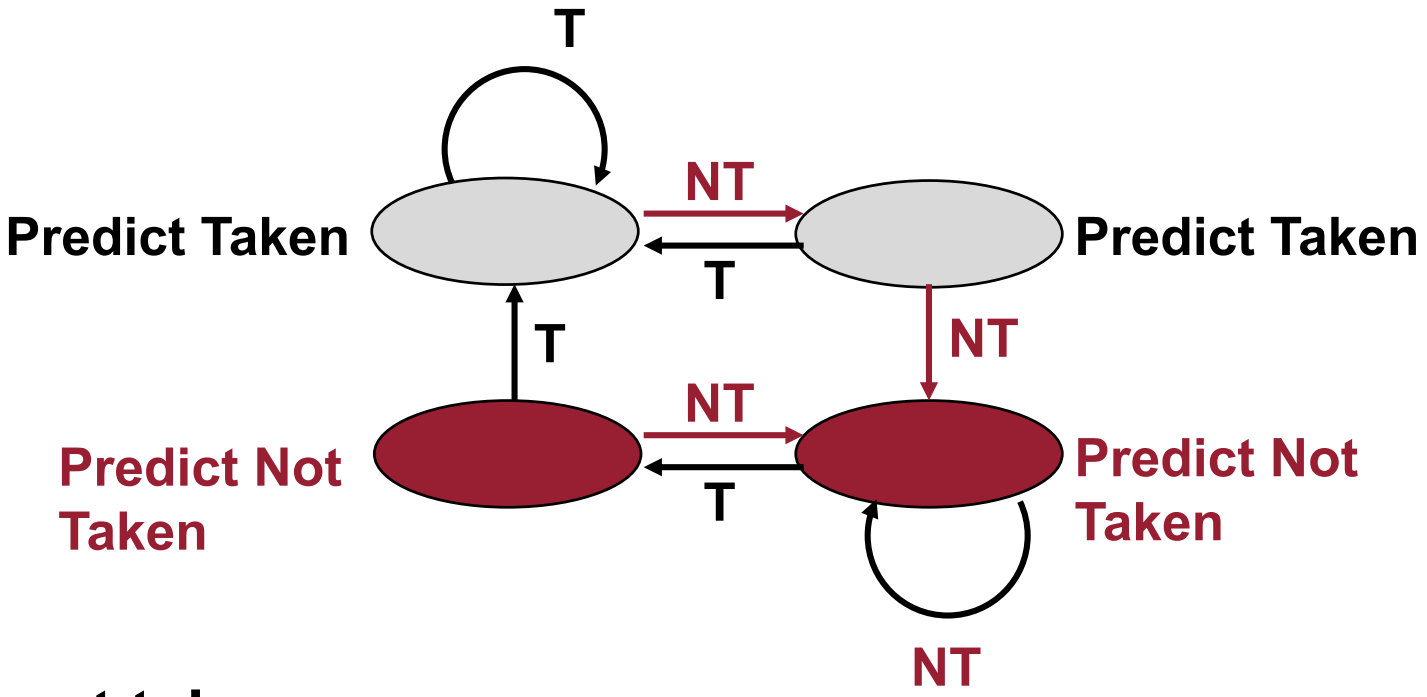
One-level Branch History Table (BHT)

- **Each branch given its own predictor state machine**
- **BHT is table of "Predictors"**
  - Could be 1-bit, could be complex state machine
- **Indexed by PC address of Branch – without tags**
  - Lower bits of the PC address are used
  - No address check (saves HW, but may not be the right address)
- **1-bit BHT keeps says whether branch was taken or not**
- **Problem: In a loop, 1-bit BHT will cause two mispredictions**
  - End of loop case: when it exits instead of looping as before
  - First time through loop on *next* time through code, when it predicts exit instead of looping
- **Solution: Use at least two bits for predictor**



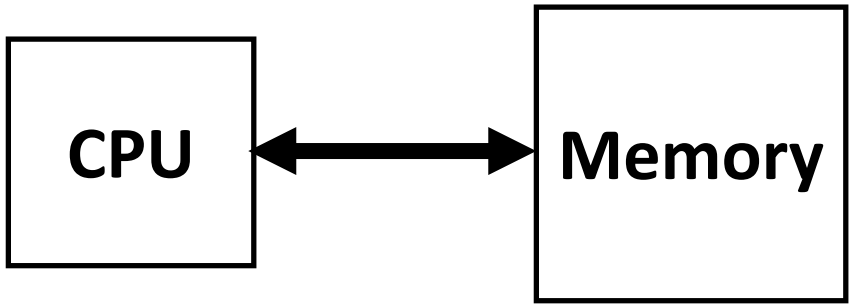
2-bit Branch Predictor

- **Solution: 2-bit scheme where change prediction only if get misprediction *twice:***



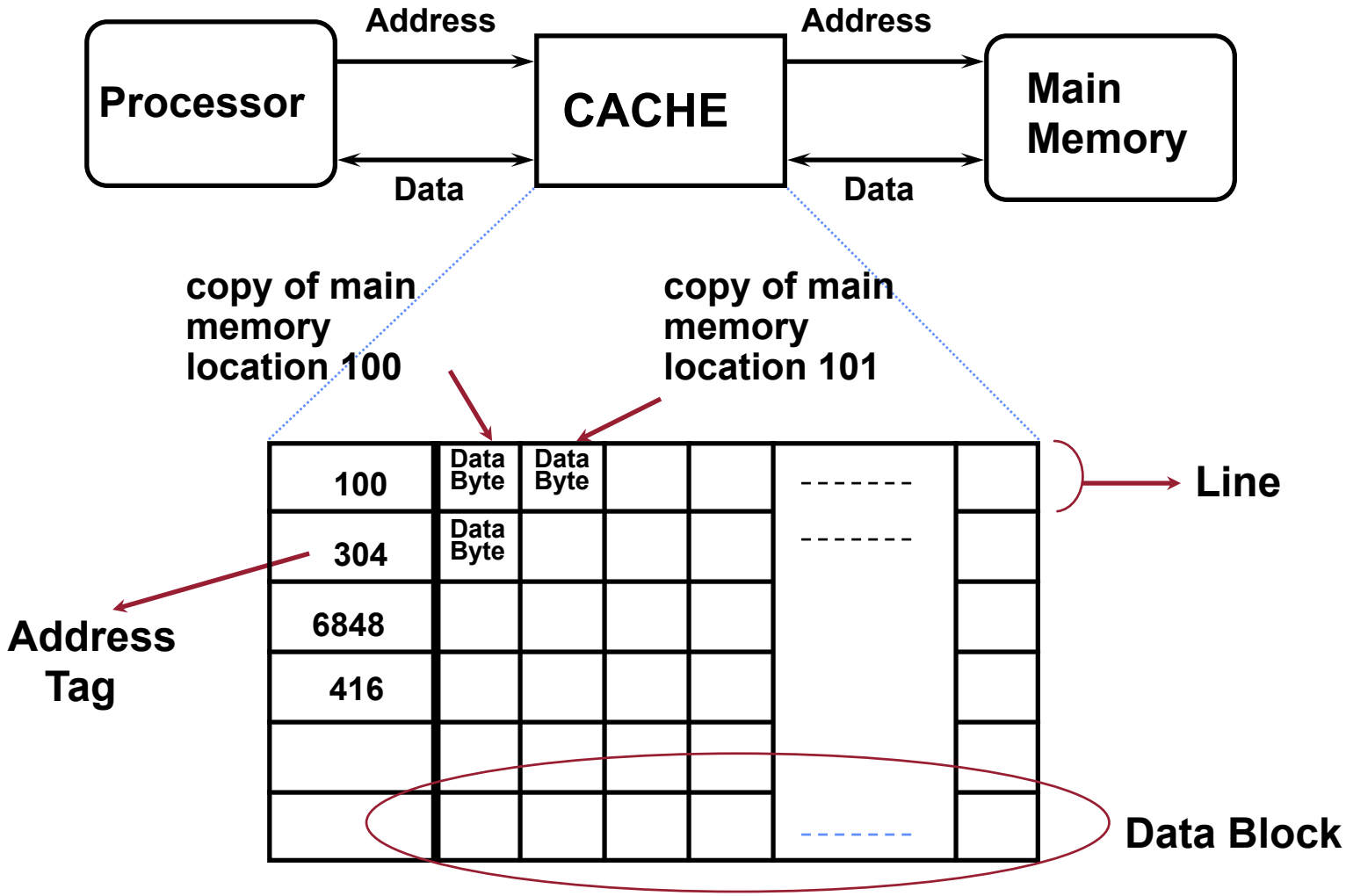
- **Red: stop, not taken**
- **Grey: go, taken**
- **Adds *hysteresis* to decision making process**

CPU-Memory Bottleneck



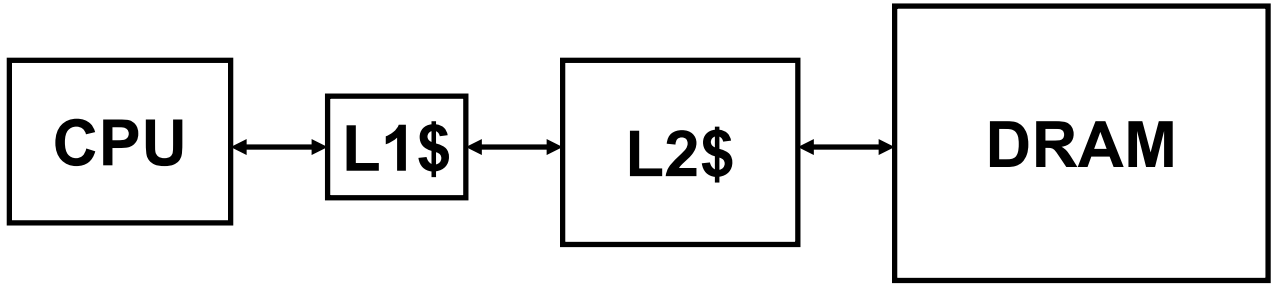
- **Performance of high-speed computers is usually limited by memory bandwidth & latency**
- **Latency (time for a single access)**
  - Memory access time >> Processor cycle time
- **Bandwidth (number of accesses per unit time)**
- **If fraction m of instructions access memory**
  - 1+m memory references / instruction
  - CPI = 1 requires 1+m memory refs / cycle (assuming RISC ISA)
- ***Also, Occupancy (time a memory bank is busy with one request)***

Inside a Cache



Multilevel Caches

- Problem: A memory cannot be large and fast
- Solution: Increasing sizes of cache at each level

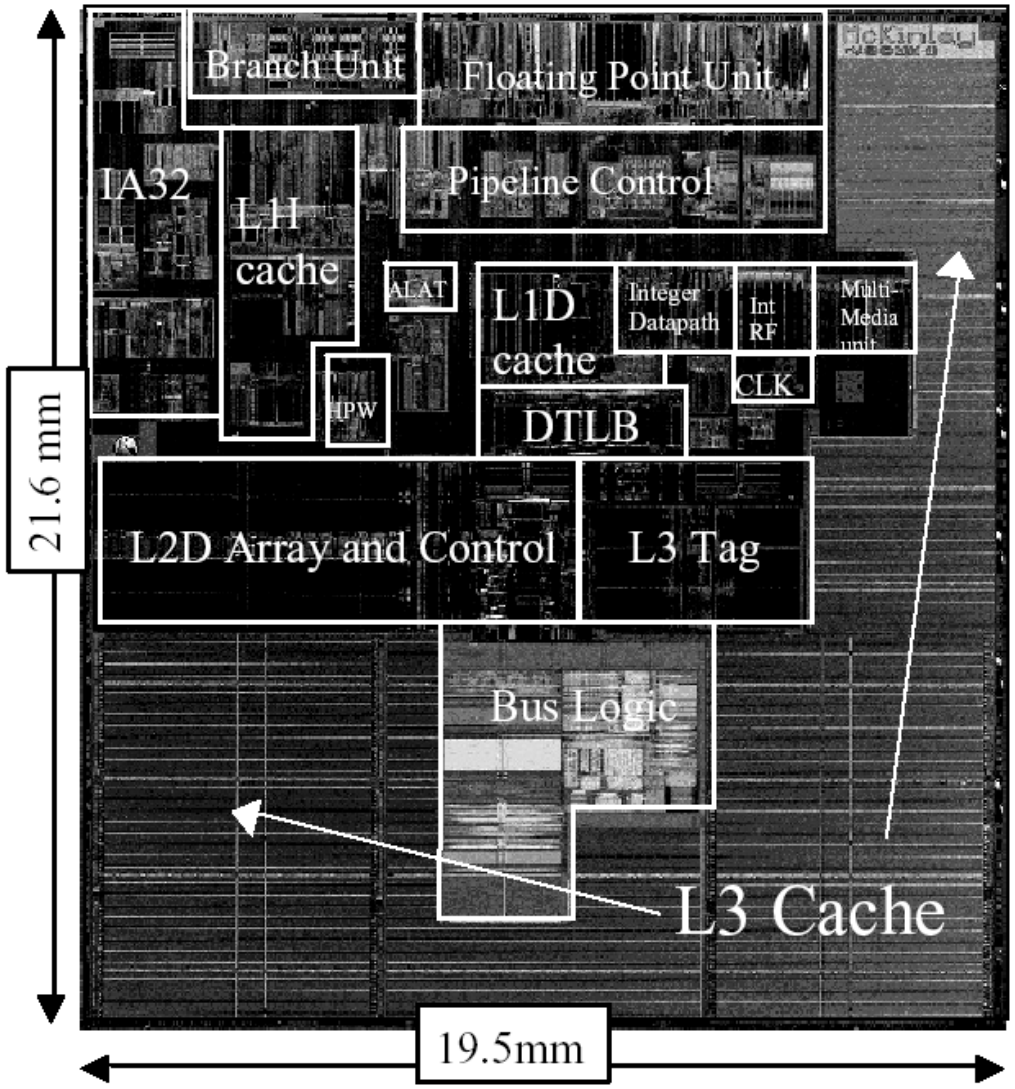


Local miss rate = misses in cache / accesses to cache

Global miss rate = misses in cache / CPU memory accesses

Misses per instruction = misses in cache / number of instructions

Itanium-2 On-Chip Caches (Intel/HP, 2002)



Level 1: 16KB, 4-way s.a., 64B line, quad-port (2 load+2 store), single cycle latency

Level 2: 256KB, 4-way s.a, 128B line, quad-port (4 load or 4 store), five cycle latency

Level 3: 3MB, 12-way s.a., 128B line, single 32B port, twelve cycle latency

Power 7 On-Chip Caches [IBM 2009]

32KB L1 I$/core

32KB L1 D$/core

3-cycle latency

256KB Unified L2$/core

8-cycle latency

32MB Unified Shared L3$

Embedded DRAM (eDRAM)

25-cycle latency to local slice