Lecture 7B

Computer Architecture II

Logic Design

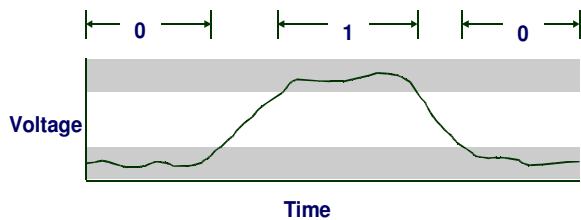---

Fundamental Hardware Requirements

- **Communication**
  - How to get values from one place to another
- **Computation**
- **Storage**

Bits are Our Friends

- **Everything expressed in terms of values 0 and 1**
- **Communication**
  - Low or high voltage on wire
- **Computation**
  - Compute Boolean functions
- **Storage**
  - Store bits of information


Digital Signals

- **Use voltage thresholds to extract discrete values from continuous signal**
- **Simplest version: 1-bit signal**
  - Either high range (1) or low range (0)
  - With guard range between them
- **Not strongly affected by noise or low quality circuit elements**
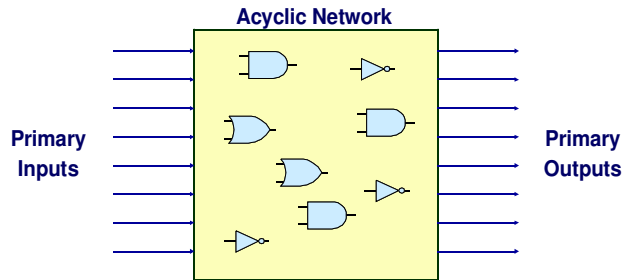 - Can make circuits simple, small, and fast

---

Computing with Logic Gates

out = a && b   out = a || b   out = ! a
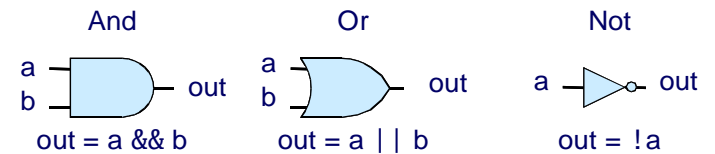
- **Outputs are Boolean functions of inputs**
- **Respond continuously to changes in inputs**
  - With some, small delay
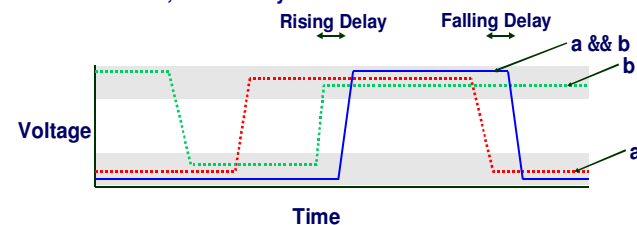
Combinational Circuits

Acyclic Network

Primary
Inputs

Primary
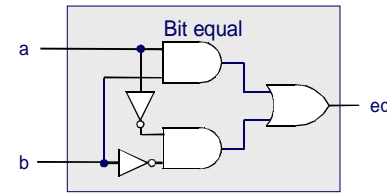Outputs

Acyclic Network of Logic Gates

- Continously responds to changes on primary inputs

- Primary outputs become (after some delay) Boolean functions
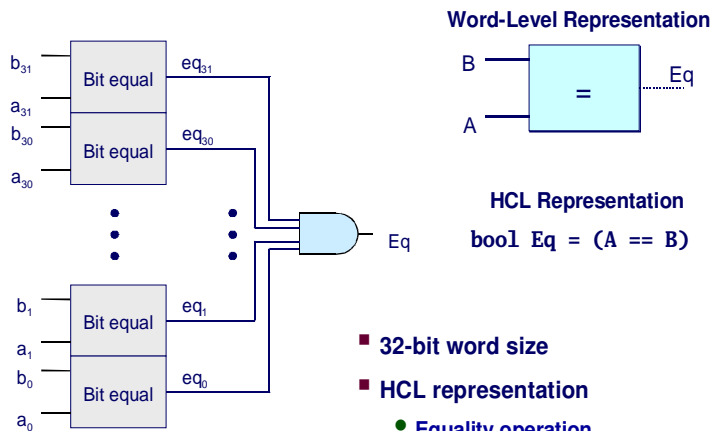 of primary inputs

---

Bit Equality

Bit equal

a

b

eq

HCL Expression

`bool eq = (a&&b)||(!a&&!b)`

- **Generate 1 if a and b are equal**

Hardware Control Language (HCL)

- **Very simple hardware description language**
  - Boolean operations have syntax similar to C logical operations
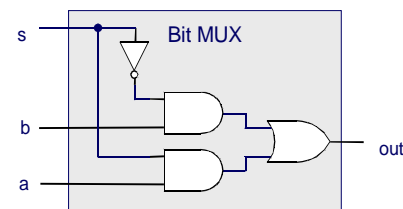- **We'll use it to describe control logic for processors**

Word Equality

b_{31}
a_{31}
Bit equal
eq_{31}

b_{30}
a_{30}
Bit equal
eq_{30}

b_1
a_1
Bit equal
eq_1

b_0
a_0
Bit equal
eq_0

Eq

Word-Level Representation

B

A

=

Eq

HCL Representation

`bool Eq = (A == B)`

- **32-bit word size**

- **HCL representation**
  - Equality operation
  - Generates Boolean value
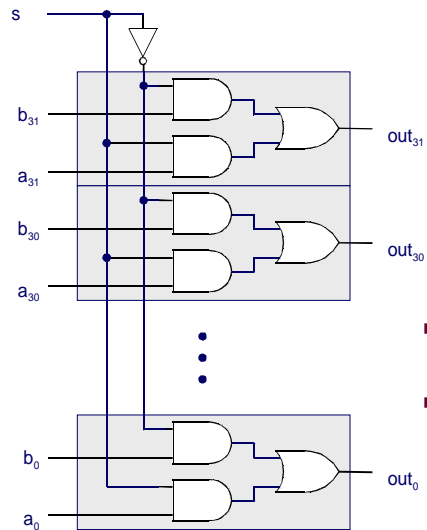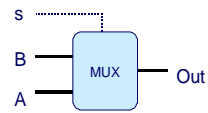

Bit-Level Multiplexor

s

Bit MUX

b

a

out

HCL Expression

`bool out = (s&&a)||(!s&&b)`

- **Control signal s**

- **Data signals a and b**

- **Output a when s=1, b when s=0**

Word Multiplexor

s

b_{31}
a_{31}

out_{31}

b_{30}
a_{30}

out_{30}

b_0
a_0

out_0

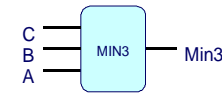**Word-Level Representation**

s

B

A MUX Out

HCL Representation

```
int Out = [
    s : A;
    1 : B;
];
```

- **Select input word A or B depending on control signal s**
- **HCL representation**
  - Case expression
  - Series of test : value pairs
  - Output value for first successful test

Datorarkitektur 2009

HCL Word-Level Examples

Minimum of 3 Words

C
B
A MIN3 Min3
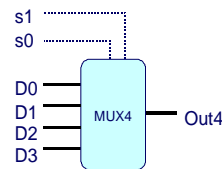
```
int Min3 = [
    A <= B && A <= C : A;
    B <= A && B <= C : B;
    1                : C;
];
```

- **Find minimum of three input words**
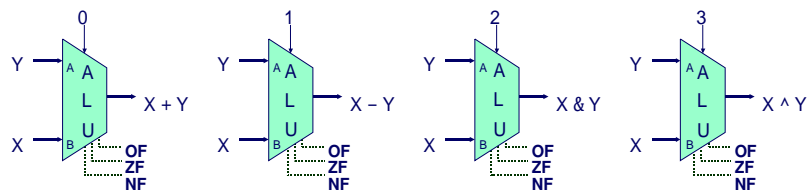- **HCL case expression**
- **Final case guarantees match**

4-Way Multiplexor

s1
s0

D0
D1
D2 MUX4 Out4
D3

```
int Out4 = [
    !s1&&!s0: D0;
    !s1    : D1;
    !s0    : D2;
    1      : D3;
];
```

- **Select one of 4 inputs based on two control bits**
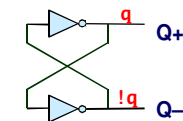- **HCL case expression**
- **Simplify tests by assuming sequential matching**

Datorarkitektur 2009

Arithmetic Logic Unit

0
Y
A
X — ALU → X + Y
B
OF
ZF
NF

1
Y
A
X — ALU → X – Y
B
OF
ZF
NF

2
Y
A
X — ALU → X & Y
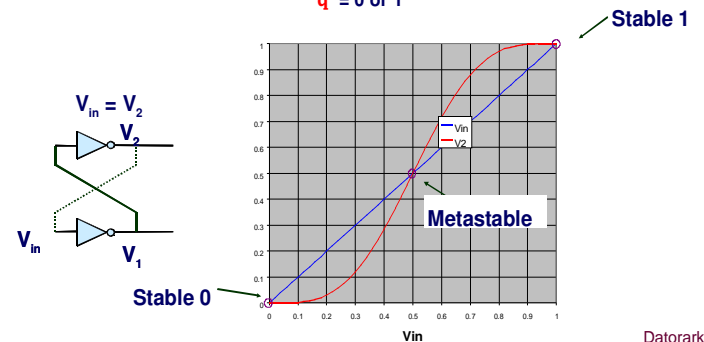B
OF
ZF
NF

3
Y
A
X — ALU → X ^ Y
B
OF
ZF
NF

- **Combinational logic**
  - **Continuously responding to inputs**
- **Control signal selects function computed**
  - **Corresponding to 4 arithmetic/logical operations in Y86**
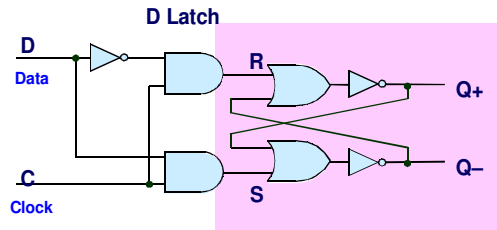- **Also computes values for condition codes**

Datorarkitektur 2009

Storing 1 Bit

Bistable Element

q

Q+

!q

Q–

q = 0 or 1

Stable 1

$V_{in} = V_2$

V_2

V_{in} V_1

Metastable

Stable 0

Vin

Datorarkitektur 2009

1-Bit Latch
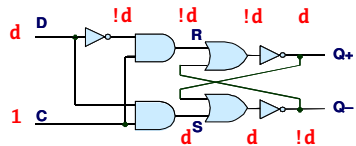
D Latch

Latching

Storing
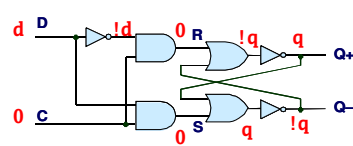
1-Bit Latch

Latching

Changing D

Time
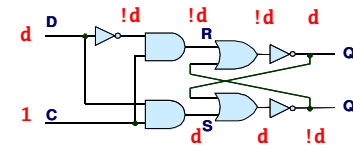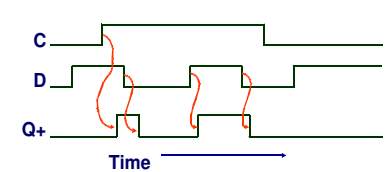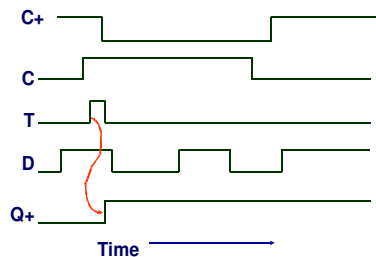
- When in latching mode, combinational propagation from D to Q+ and Q–
- Value latched depends on value of D as C falls

Edge-Triggered Latch
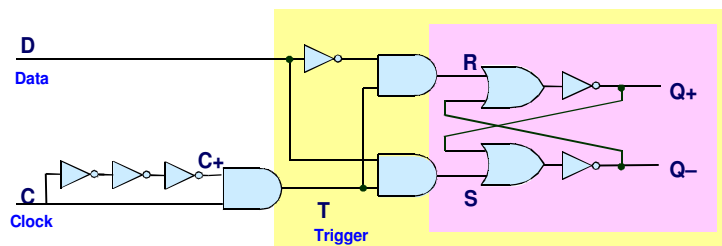
- Only in latching mode for brief period
  - Rising clock edge
- Value latched depends on data as clock rises
- Output remains stable at all other times

Time

Registers

Structure
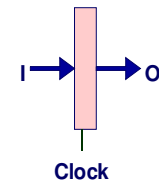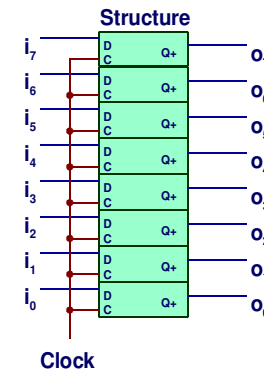
Clock

- Stores word of data
  - Different from *program registers* seen in assembly code
- Collection of edge-triggered latches
- Loads input on rising edge of clock

Register Operation

State = x

Input = y   Output = x   **x**

Rising clock

State = y

Output = y   **y**

- **Stores data bits**
- **For most of time acts as barrier between input and output**
- **As clock rises, loads input**

State Machine Example
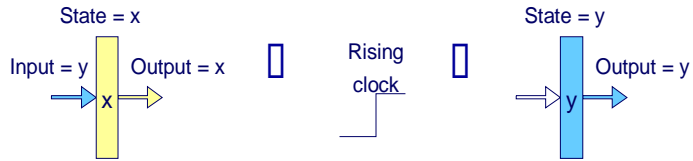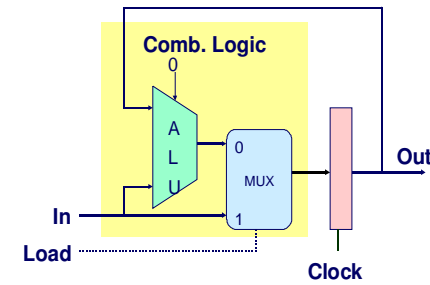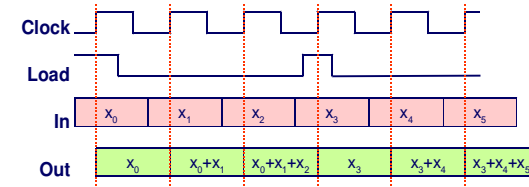
Comb. Logic

0

A L U

0 MUX 1

Out

In

Load

Clock

- **Accumulator circuit**
- **Load or accumulate on each cycle**

Clock

Load

In x_0 x_1 x_2 x_3 x_4 x_5

Out x_0 x_0+x_1 $x_0+x_1+x_2$ x_3 x_3+x_4 $x_3+x_4+x_5$

Random-Access Memory

valA

srcA   A

Read ports   **Register file**   W   valW   dstW   Write port

valB

srcB   B

Clock
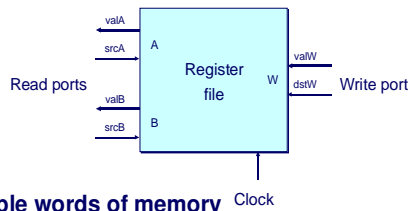
- **Stores multiple words of memory**
  - **Address input specifies which word to read or write**
- **Register file**
  - **Holds values of program registers**
  - **%eax, %esp, etc.**
  - **Register identifier serves as address**
    - » ID 8 implies no read or write performed
- **Multiple Ports**
  - **Can read and/or write multiple words in one cycle**
    - » Each has separate address and data input/output

Register File Timing

valA

srcA   A   2   **x**

Register file

x
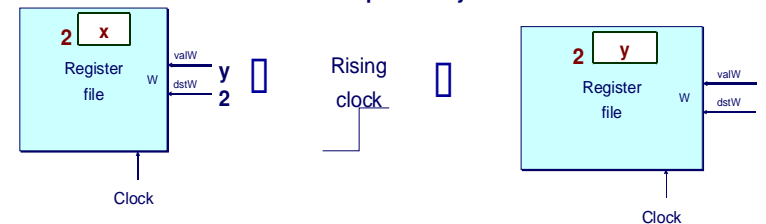
valB

srcB   B

2
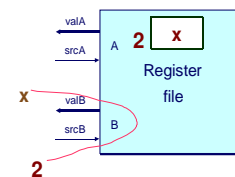
Reading
- **Like combinational logic**
- **Output data generated based on input address**
  - **After some delay**

Writing
- **Like register**
- **Update only as clock rises**

2   **x**

Register file   W   valW   dstW   y   2

Clock

Rising clock

2   **y**

Register file   W   valW   dstW

Clock

Hardware Control Language

- **Very simple hardware description language**
- **Can only express limited aspects of hardware operation**
  - **Parts we want to explore and modify**

Data Types

- `bool`: **Boolean**
  - **a, b, c, ...**
- `int`: **words**
  - **A, B, C, ...**
  - **Does not specify word size---bytes, 32-bit words, ...**

Statements

- `bool a = `*`bool-expr`*` ;`
- `int A = `*`int-expr`*` ;`

HCL Operations

- **Classify by type of value returned**

Boolean Expressions

- **Logic Operations**
  - `a && b, a || b, !a`
- **Word Comparisons**
  - `A == B, A != B, A < B, A <= B, A >= B, A > B`
- **Set Membership**
  - `A in { B, C, D }`
    - » Same as `A == B || A == C || A == D`

Word Expressions

- **Case expressions**
  - `[ a : A; b : B; c : C ]`
  - **Evaluate test expressions a, b, c, ... in sequence**
  - **Return word expression A, B, C, ... for first successful test**

Summary

Computation

- **Performed by combinational logic**
- **Computes Boolean functions**
- **Continuously reacts to input changes**

Storage

- **Registers**
  - **Hold single words**
  - **Loaded as clock rises**
- **Random-access memories**
  - **Hold multiple words**
  - **Possible multiple read or write ports**
  - **Read word when address input changes**
  - **Write word as clock rises**