Computers from NAND gates.

Build

\[ \text{DFF} = \text{DFF} \]

\[ \text{DFF} = \text{DFF} \]

\[ \text{DFF} = \text{DFF} \]

\( \text{D} \rightarrow \text{move DFF} \)

\( \text{D} \rightarrow \text{move latch} \)

\[ \text{D} \]

\[ \text{DFF} = \text{DFF} \]

\[ \text{DFF} = \text{DFF} \]

\[ \text{DFF} = \text{DFF} \]

\( \text{D} \rightarrow \text{move latch} \)

\[ \text{D} \]

\[ \text{DFF} = \text{DFF} \]

\[ \text{DFF} = \text{DFF} \]

\[ \text{DFF} = \text{DFF} \]

\( \text{D} \rightarrow \text{move latch} \)

\[ \text{D} \]

\[ \text{DFF} = \text{DFF} \]

\[ \text{DFF} = \text{DFF} \]

\[ \text{DFF} = \text{DFF} \]

\( \text{D} \rightarrow \text{move latch} \)
Full Adder

\[
\begin{align*}
\text{in} & = & \text{out} \\
\text{in} & = & \text{carry}
\end{align*}
\]

A full adder makes the 1-bit adder.

\[
\begin{align*}
y_2 & = x_2 \\
\bar{y}_2 & = x_2 \\
\bar{y}_1 & = x_1 \\
\bar{y}_0 & = x_0 \\
o & = 0
\end{align*}
\]

MUXES

\[
\begin{align*}
A^{(0)} & \quad \text{MUX} \\
B^{(1)} & \quad \text{sel} \\
\text{out} & \\
\end{align*}
\]

\[
\begin{align*}
A & \quad A \quad D \\
B & \quad B \quad \text{sel}
\end{align*}
\]

COUNTS

\[
\begin{align*}
x_1 & \quad \text{MUX} \\
\text{sel} & \quad \text{select}
\end{align*}
\]

\[
\begin{align*}
\text{Reg} & \quad \text{CLK} \\
\text{select} & = 0 \quad \text{load} \\
\text{select} & = 1 \quad \text{increment}
\end{align*}
\]
3-bit JK Racer Oute

![Diagram]

- **States**: Mealy

- **Input**: I/p

- **Logic**: Output

- **New State**: \( \text{out} = f(I/p, \text{state}) \)

- **Output**: \( \text{out} \) on edges

- **States**: \( I/p \) in states
Moore is synchronous

Grey code: only 1 bit changes each step

Reflected Grey code (Minimal method)

\[ Q_2 = Q_0 Q_1 Q_0^* \]

\[ Q_0 = D_2 D_1 + D_2 D_1^* \]

\[ Q_1 = \]

\[ Q_{1*} \]

\[ Q_{z*} \]

\[ Q_{z} \]

\[ Q_0 \]

\[ Q_1 \]

\[ Q_2 \]
\( \begin{array}{ccc}
0 & 0 & 0 & 0 & 0 & \text{DUAL} & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 1 & \rightarrow & 0 & 1 & 1 & 1 \\
0 & 1 & 0 & 0 & \text{ } & 1 & 0 & 1 & 0 \\
0 & 1 & 1 & 0 & \text{ } & 1 & 1 & 0 & 0 \\
1 & 0 & 0 & 1 & \text{ } & 1 & 0 & 0 & 1 \\
1 & 0 & 1 & 1 & \text{ } & 1 & 1 & 1 & 0 \\
1 & 1 & 0 & 0 & \text{ } & 1 & 1 & 0 & 0 \\
1 & 1 & 1 & 0 & \text{ } & 1 & 1 & 1 & 0 \\
\end{array} \)

**Back to Grey counter:**

\[ a_2 a_1 a_0 a_3 \quad J K \]

\[ \begin{array}{ccc}
0 & 0 & 0 & a_1 & 1 & 1 & \rightarrow & \text{make Kmap} \\
0 & 0 & 1 & 1 & 1 & 0 \\
0 & 1 & 0 & 0 & 0 & x \\
0 & 1 & 1 & 0 & 1 & x \\
1 & 0 & 0 & 0 & 0 & x \\
1 & 0 & 1 & 0 & 1 & x \\
1 & 1 & 0 & 0 & 1 & x \\
1 & 1 & 1 & 0 & 0 & x \\
\end{array} \]

\[ K_0 = \overline{a_2} a_1 a_0 + a_2 \overline{a_1} \]

**State Assignment**

Counters = trivial

But

\[ \begin{array}{cccc}
& x & 0 & 1 & 1 \\
x & 0 & 1 & 0 & x \\
\end{array} \]

\[ a_0 a_2 a_1 a_0 \]

assign states to:

- make o/p loop as short as possible
- make initial state
- make adjacent states as simple as possible
Given $\frac{3}{1} \times \frac{17}{10}$
XNOR

3-State Latch

RS Latch

Debounced Switch

A

Delay

\[ \text{in} \Rightarrow \text{out} \]

\[ \text{in} \Rightarrow \text{out}, \text{then goes to} + t \text{ gate delay, let it zero} \]

in

out

3 gate delays
Timing Analysis

Flip Flops:
- Max delay: 60 ns
- Min delay: 30 ns
- Set-up time: 10 ns
- Hold time: 10 ns

→ from i/p CLK changing to o/p = changing.

CLK

→ time i/p's must be unchanging before clock change.
→ time unchanging after clock.

CLK

PLA

\[ X = \overline{A}B + BC \]

\[ Y = AC + BC \]
ROM all patterns decoded

A

B

C

0 1 2 3 4 5 6 7
Virtual Memory

Instruction Set Design
Processor Implementation

Hardware Tutorial

Caches

Virtual Memory

I/O

CPU

CACHE

Memory

DRAM ~ 100ns

Imperial Locality
Spatial Locality

1024 bytes
32 lines

16 bytes = 4 32 bit words!

Can Address about many of virtual/physical
\[ v_0 \{ w_0, w_1, w_2, w_3 \} \quad R_1, R_2, \ldots \]

\[ q \leq 1K \text{ of data (not incl. the tag & dirty bits),} \]

\[ v \rightarrow \text{valid bit} \quad \text{not valid} \]

\[ w \rightarrow \text{dirty bit} \]

\[ \text{Write through} \rightarrow \text{goes to memory immediately (don't need the extra bit)} \]

\[ \text{fast on a cache miss} \]

\[ \text{Write back (copy back)} \]

\[ \text{Cache doesn't write to memory} \]

\[ \text{Stalls on misses!} \]

\[ \text{adv. don't need to write each time} \]

How do you decide which to throw away?

1. Random (fairly)

2. LRU (replace LRU)

Random - for HW caches, almost as good! Why?

LRU - don't want to do each access.

no D bit in write through cache.

Direct mapped \( \rightarrow \) write back yes.

in the other case no.

Write through \( \rightarrow \) maybe yes/no.

Direct mapped \( \Rightarrow \) no.

Valid bit always.

Dirty bit never sometimes.

Replacement bit sometimes never.
**Virtual Memory**: 

32 Address

- Band
- Silence the translation
- Need memory to do address translation

- Pool
- More memory
- Multiple processes can start at zero
- Memory protection

Virtual Address translation can occur in one of two processes.

Could physically access virtual addresses, which is a problem - don't want to have shared virtual addresses.

So, don't need to do the address translation for each.

Can't do virtual mem 32 add translation!

32 bit address

<table>
<thead>
<tr>
<th>Page</th>
<th>Page Offset</th>
</tr>
</thead>
</table>

8SD 4.3

4K byte page size

12 bits

Translation

Protection protocols

Page table

(can get large)

Protection

Page Table

Protocols

PTE

Page Table Buffer

Read only cache

Changes only when page Table
page fault;

now no program ever needs entire page table
VAX: page the page table itself
bits in virtual memory

now you need two memory accesses

page table hangs out in system space
→ protection

Tree of pages

→

but say keep going or stop!

segment: variable length pages
I/O: new memory mapped

e.g., any address in ffff8000 - ffff8010 will go to

Disk Drive

Could be virtual addresses

How do you tell the device is ready
(i) poll: check each exit place every now and then
status bit

(ii) interrupts

3 ways to actually comm.
(i) Programmed I/O: CPU actually goes out

for slow devices

(ii) DMA: disks, video, high-speed stuff

CPU builds table

Addr of. virtual mem address for table

entry in memory but not in physical!

4 pages

(iii) Data channel

Bunch of processes share an I/O processor

Bunch of I/O devices heavy off a single DMA processor
Wish to design an instruction set.

3.2

Word/Address

How many addresses in each instr.

0
1
2
3

\[ \text{ADD} \quad \text{ADD} \quad \text{ADD} \]

2 sources & 1 destination

\[ \text{ADD} \quad \text{ADD} \]

implicitly use 7

Accumulator

0 \Rightarrow \text{status based}

Addressing Modes: out these days

\( \Rightarrow \text{a subset of instr.} \)

(specific by shift and)

Now = Load/Store

Registers file size \( \Rightarrow 32 \) is a popular\*

Floating pt. vs. int. registers

Assns vs data registers

Before: wanted to encode more cheaply (arbitrary).

Now not a real issue

so don't need separate addr & data reg.
Instruction
- ADD
- SUB
- MUL
- DIV/REM
- AND
- OR
- XOR

Branch/Jump:
- src -> branch on cond
- src -> src negate cond

VAX
- \text{opcode} \quad \text{some specify immediate}
- \text{very little space, slow}
- decode

MIPS
- \text{one ins. per cycle}
- \text{very fast}
- \text{every instr. is 32 bits wide!}
- \text{one of 4 instr. formats,}

\text{IMPLEMENTATION}

\text{FETCH}
- (fast on load/store)

\text{DECODE}

\text{EXECUTE}

\text{control}
USE LOTS OF REGISTERS

To latch everything:

1. If the user puts the HL register in the latch register
2. If the user puts the DE register in the latch register
3. If the user puts the BC register in the latch register
4. If the user puts the AF register in the latch register

for R6/7

1. If R6 = R7, then latch
2. If R7 = R7 + 1, then latch
3. If R6 = R7, then latch
4. If R7 = R7 + 1, then latch
node wasteful, lot of zeroes

Makes sense to collapse ALU ops down to decisiveness

can narrow further

hole a separate line of code for AND, +, OR etc
even though everything anything same

2 level mode

Hardware Control

for same above each