CS 251 - S23
Computer Organization and Design
Full Course Notes

With Prof Zille Huma Kumal

In-class notes were tricky in this class. I didn't write everything, but I hope this helps.

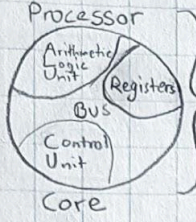
Josiah Plett

CS 251 ①

Branching: 4. B #1 → 5.

64-bit architecture

① Introduction + ARM



WHAT ISA A COMPUTER?

Instruction **S**et **A**rchitecture
↳ Basically the API for the hardware that your core can execute

Computer design: the choice of ISA

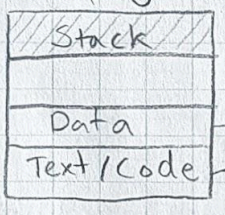
Computer architecture: ISA + computer organization

HOW COMPUTERS WORK
- User Program
- High Level Language
- Assembly
- Operating System
- Machine Language
- Control Units

ARMv8 → LEGv8

ARM is our ISA

ARM Program / Mem



② ARM Simulations + Performance

WORD: 4 bytes. ← INSTRUCTIONS

Memory
- Register
- RAM
- Hard disk

GPR: general purpouse register.
XZR: zero.
PC: program counter (SPR)
SPR: special purpose register.
#: immediate. -256 → 255.

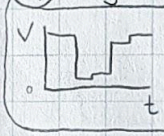
③ Performance

CPU Time: # of clock cycles X clock cycle time

Clock Rate: 1 / clock period

Generally, we count total instructions.

④ Digital Logic Design



Karnaugh alternative: Don't Cares

⑤ Boolean Logic

XXX ☺

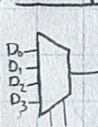
⑥ Logic Gates

ONLY NAND
- SOP form
- invert between

XOR

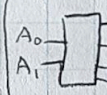
1 if odd # of 1's

Multiplexor



2^n lines
n select lines
output: D_S

Decoder



n inputs
2^n outputs

ALU

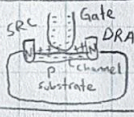
Arithmetic-Logic Unit

PLA

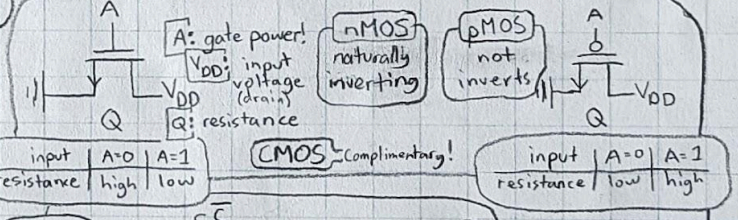
Programmable Logic Array

⑦ Transistors, Clocks, CMOS

Metal Oxide Semiconductor Field-Effect Transistor — Intestable material!



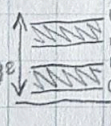
current is drain to src



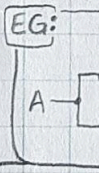
input resistance	A=0	A=1
	high	low

CMOS - Complimentary!

input resistance	A=0	A=1
	low	high

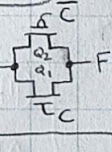


EG:

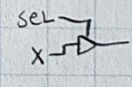


A	Q_1	Q_2	F
0	L	H	1
1	H	L	0

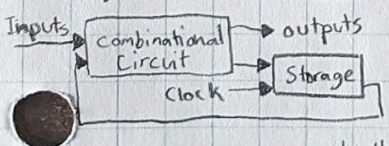
3-state buffer Gate



C	X	Q_1	Q_2	F
0	0	H	H	Z
0	1	H	H	Z
1	0	L	L	0
1	1	L	L	1

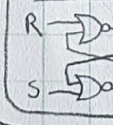


⑧ Sequential Logic



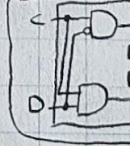
conceptually understand that

SR Latch (NOR)



S	R	Q	$\bar{Q}$	
0	0	Q	$\bar{Q}$	• Latch (no change)
0	1	0	1	• Reset state
1	0	1	0	• Set state
1	1	?	?	• Undefined

D-Latch



C	D	Q^{t+1}	
0	0	Q^t	
0	1	$\bar{Q}^t$	
1	0	0	(reset)
1	1	1	(set)

Instruction Formats

R-Format (register)
- 3 operands, in registers
- Store in first register

ADD X3, X1, X2

D-Format (data, %8)
- 3 operands, 2 in GPR

LDUR X1, [X2, #24]
X1 = (X2 + 24)* (↳ X2[3])

STUR

I-Format (immediate)
- 3 operands, 1 is #

SUBI X1, X2, #100

B-Format (branch)
- unconditional goto

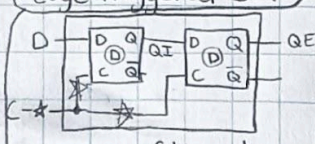
B #28 (word offset)
PC = PC + 4×28

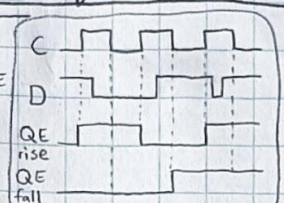
CB-Format (conditional)
- conditional goto

CBNZ X1, #8
(not zero; ↳ compare to 0)

(9) Edges, DFF, Sequential Circuits

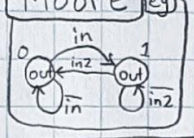
Edge-triggered DFF





Total Transistors: $46 = 2\times((4+6)\times2+2)+2$

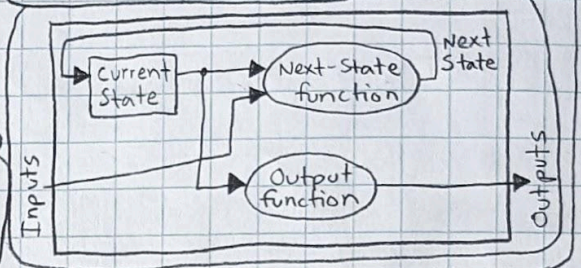
Moore eg



Next State Table

S	EWcar	NScar	S'
00	0	X	01
01	1	X	10
10	X	0	11
11	X	1	00

Finite-State Machines



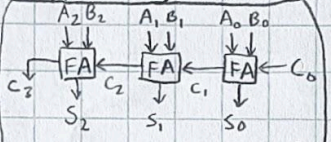
(10) Data Representation/Manipulation. 1-Bit ALU

Two's Compliment!

Full Adder

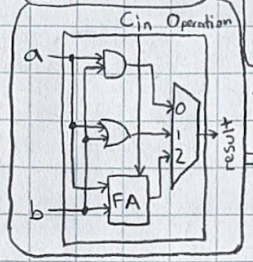
C_{in}	A	B	C_{out}	Sum
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Ripple-Carry Adder



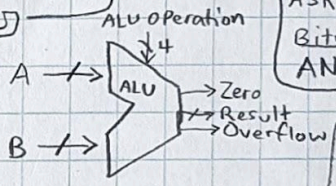
To implement subtract:
- Not all the B_ns.
- Set $C_0 = 1$.

1-Bit ALU



IMPROVEMENTS

- Mux a and b for $\overline{a}$ and $\overline{b}$



ARM instructions

LSL - logical shift Left
ASL - Arithmetic shift Left
LSR - 0's on left
ASR - sign on left
Bitwise:
AND, OR, EOR (NOT=EOR w/1)

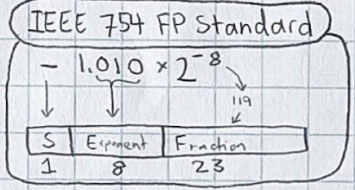
(11) Floating Point (FP)

Normalized: fraction starts w/1.

Overflow: exponent too big.

Underflow: exponent too small.

IEEE 754 FP Standard



Special cases

E	F	Value
0	0	0
all 1	0	$\pm\infty$
all 1	non-0	NaN

Bias notation

bias = 127

$2^{-8} \rightarrow E = 119 - bias$

(12) Single Cycle Processors

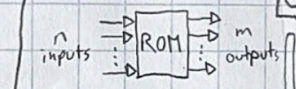
Byte addressing: either LSB or MSB, in ARM

Primary Storage/State elements

ROM Registers SRAM DRAM

- DRAM slower + cheaper than SRAM
- Refresh controller must allow read/write access
- Possibility of getting more bites (page-mode RAM)

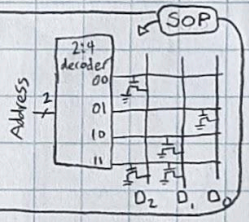
ROM



2^n m-bit words

eg

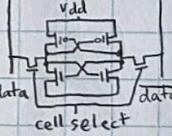
$A_1 A_0$	$D_2 D_1 D_0$
00	011
01	110
10	101
11	001



RAM

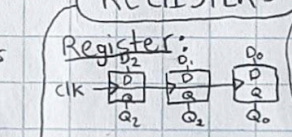
SRAM

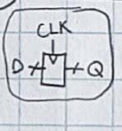
Cell:



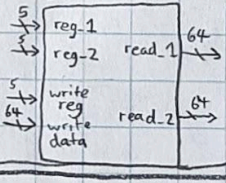
REGISTERS

Register:



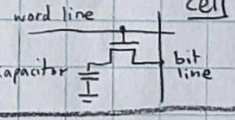


Register File:



Shared clock: all change at once!

DRAM



(13) Single Cycle Processor Implementation

Future josiah: read the slides and remember the "high level" ideas.

(14) ALUs

ALUOp:

00	Add	R-format
01	pass B	CB-format
11	R-format	

CS 251 (2)

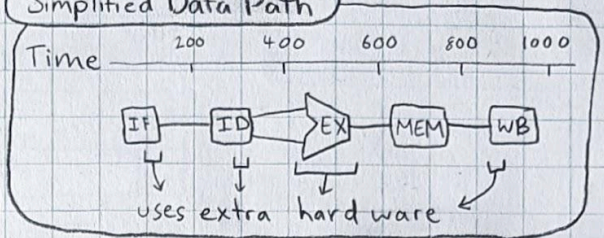
(15) Pipelining

Pipelining: Parallel execution of instructions.

Datapath Stages

1. IF Instruction Fetch — fetch instruction.
2. ID Instruction Decode — read regs & decode.
3. EX Execute — execute operation.
4. MEM Memory — get operand in datamem.
5. WB Write Back — write result to a reg.

Simplified Data Path



NOP: no operation

"forwarding" is feeding EX into next ID, for example.

"stalling" waiting another step in case it's impossible

Data Hazard: result of an instruction is needed by another.

Structural Hazard: hardware can't support the combo we need.

Control Hazard: CB instruction might change next instructions.

→ control hazard solutions:
1. Stall
2. Predict ← "dynamic" uses past prediction behaviour.
3. Delay Decision ← more unaffecting instr to after

Pipeline Registers store intermediate values between clock cycles.

(16) Forwarding

Forwarding: to the EX stage
- Pass ALU Result from a later pipeline back into EX forwarding unit.

Notation? IDK, see slides

Needed When: (both 1 and 2)
1. RegWrite = 1 in MEM or WB.
2. Destination is not X31

EX hazards

MEM hazards

(17) Contol Hazards

We have a Hazard Detection Unit that stalls if a hazard is detected.
↳ set all controls to zero, esspecially (mem read, mem write, regwrite.)

Load-Use Hazard: LDUR isn't fast enough!

Flush: Don't run this instruction again!

CBZ: doesn't need ALU!

Branching Types

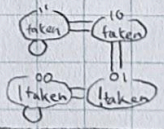
Branch Not Taken: assume it doesn't branch (always).

Branch Taken: assume it does branch, always.

Dynamic Branch Prediction

1-bit: use a bit to remember if branched last time. 1 = was taken. __init__ = ??

2-bit: use the DFA



(18) Code Rearrangement

Code Rearrangement: swap lines of arm code to remove hazards.

Loop unrolling: remove values only used for CB instructions and take care of the looping yourself! (bad for caching).

Notes: STUR should be unchanged. Don't swap STUR↔LDUR. Use branch-delay slots instead of flushing. Don't swap forward or backward dependencies. Don't move instructions in/out of a loop.

Exceptions

- External event (eg I/O device) changes control flow.
- Internal event (eg overflow) changes control flow.

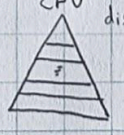
Solution ☑ save PC for interrupted instruction, service the instruction.

ELR: Exception Link Register

ESR: Exception Syndrome Register

(18 20) Caching

Cache: probably SRAM. Memory but on the chip!



Block: minimum info unit

Hit: info present when requested

Miss: info not present, must be copied up.
↳ stalls processor.

Direct-Mapped-Cache:

to differentiate: store "tag" (modded-out bits) in data itself.

(19) More Cache

Block-size Read/Write

Just read/write blocks at a time!

V = valid bit: iff has right value!
D = dirty bit: iff block's been changed.

2-way Set-associative Cache

- Look at LSB's for the tag
- Have 2 values in each spot (tag)!

↳ Fully associative: maximall "#-way"

Caches search tags in parallel.

(20) More Cache

When cache is full? What to evict?
LRU: least recently used. ← needs to track order.
↳ "use" bit → the lru word.

Index: General slot (from LSB) for an address.
Tag: Differentiator for associative
Linesize: Number of items in each row.

AMAT – Average Memory Access Time

AMAT = Time for hit + Miss Rate × Miss Penalty.

★ I didn't write any real notes on page tables or TLBs, so I recommend going through the tutorials/lectures for that.

Tag		Index		Block		Byte	
63	...		5	4	3	2	1	0

A 2^n-way cache of size 2^k bytes with a block size of 2^b:

- 2 byte-bits
- b block-bits (linesize = bytes × blocks)
- index bits $i = k-2-b-n = k-1s-n$
- tag bits remain $64-2-b-i = 64-1s-i$

Page Replacement Schemes

LRU: Least Recently Used
LFU: Least Frequently Used
FIFO: First In First Out
Random: Random ☺

(21) Virtual Memory

Virtual Memory: between main memory and secondary storage (disks). (non-physical).

Page: virtual memory block

Page Fault: virtual memory miss

USED TO
- share memory between programs
- allow single user to exceed main mem size

Process: running program

Time Slice: block of time each process gets to run

LEC 22

Page Table: virtual page → physical memory

TLB: Translation Lookaside Buffer

Stores translations between virtual and real addresses.
(TLB is a cache specifically for translations.)

↑ my ideal daily routine ↑

↓ everyone happily studying for 251 together ↓