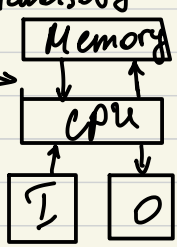
06/09

A brief history

1. Pre history (1700 - 1930s)
   Pascal, Leibniz — calculators
   ↳ binary system (1705)
   Jacquard — weaving machines (end of 18th c.)
   Johann Helfrich von Müller (1786
   Charles Babbage → Analytical Engine (1837)
   ↳ concept: arithmetic unit + memory + I/O devices + IFs conditionals
   Ada, countess of Lovelace → first programmer (1840s)
   Vannevar Bush (1931) — mechanical integrators (analog computers)

2. I generation: electro-mechanical (1930s - 1960s)
   ASCC - Howard Aiken 1937 - 1944)
   ENIAC - John Mauchly + John Presper Eckert (1943-1947)
   first computer IP lawsuit: vs John Atanasoff
   EDVAC - Mauchly + Eckert (1948 - 1949)
   basis of Von Neumann Architecture →



3. II generation: transistors (1965 - 1975)
   DEC PDP-1 (1959)
   Cray's CDC 6600 - first supercomputer

   Intel 4004 (1971)
   Apple (1978)

   first monitor (1951)
   first mouse (1968)

IBM personal computer – released in 1980s
open standards to third-party

4th microprocessors (1960s – today)

4. 4th generation: multi-computing
ARPANET (1965-1969) – Leonard Kleinrock
1972 – ARPANET public + Email

08/09

Assembly

Logic circuits [3, 4]
Data representation [5, 6]
Instruction Set Architectures [7, 8]
Software [2]
- assembly
- C
- C++ / Java

13/09

Logic Circuits 1

$x \cdot x = x^2 = x$
$x \wedge x = x$

$f(x) = a_0 + a_1 x + a_2 x^2 + \dots$
$= a_0 + a_{n-1} x$

$x + x = 2x = x$
$x \vee x = x$

$f(x) = a \cdot x + b \cdot (1 - x)$
$f(x) = f(1) \cdot x + f(0) \cdot \bar{x}$

$1 - x = \bar{x}$
$\neg x$

$f(x,y) = f(1,1) \cdot x \cdot y + f(1,0) \cdot x \cdot \bar{y} +$ minterms
$f(0,1) \cdot \bar{x} \cdot y + f(0,0) \cdot \bar{x} \cdot \bar{y}$

XOR

x	y	$x \otimes y$
0	0	0
0	1	1
1	0	1
1	1	0

$$f(x,y) = f(1,1)\cdot x\cdot y + f(0,1)\cdot \bar{x}\cdot y + f(1,0)\cdot x\cdot \bar{y} + f(0,0)\cdot \bar{x}\cdot \bar{y}$$

$$x \otimes y = f(0,1)\cdot \bar{x}\cdot y + f(1,0)\cdot x\cdot \bar{y}$$

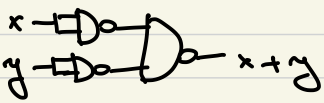
$$x \wedge y = f(1,1)\cdot x\cdot y$$

$$f(x,y,z) = \overline{(\bar{x}\cdot\bar{z})}\cdot y + \bar{z}\cdot\bar{x} = (x+z)\cdot y + \bar{z}\cdot\bar{x} = xy + zy + \bar{z}\cdot\bar{x}$$

$$\overline{x\cdot y} = \bar{x} + \bar{y}$$

$$\overline{x+y} = \bar{x}\cdot\bar{y}$$

x	y	z	f
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1



x	y	f
0	0	0
0	1	1
1	0	1
1	1	1

$$f = x\cdot y + \bar{x}\cdot y + x\cdot\bar{y}$$

$$f = \bar{x}\cdot\bar{y}\cdot\bar{z} + \bar{x}\cdot\bar{y}\cdot z + \bar{x}\cdot y\cdot z + x\cdot y\cdot z$$
$$= (\bar{x}\cdot\bar{y})(\bar{z}+z) + y\cdot z(\bar{x}+x)$$
$$= \bar{x}\cdot\bar{y} + y\cdot z$$

z \ xy	00	01	11	10
0	1	0	0	0
1	1	1	1	0

w \ xy	00	01	11	10
00	1	1	0	1
01	1	1	0	1
11	0	0	0	0
10	1	1	1	0

$$f = \bar{x}\cdot\bar{z} + \bar{y}\cdot z + yz$$

z \ xy	00	01	11	10
0	1	1	0	0
1	1	1	1	1

15/09 Logic Circuits 4

20/09 Data Representation 5

$\mathbb{Z}^+$ → nonnegative integers

$\Sigma = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$

2557_{10} → binary (radix-2)

2 5 5 7

0010 0101 0101 0111

-52_{10} 8 bit 2's comp

-64_{10} → 1000 0000

12_{10} → 1000 1100

↳ 1100

$52_{10} \rightarrow 00110100$

↓

$-52_{10} \rightarrow 11001100$

4998_{10} in BCD

4998:2 = 2499	0	1
2499:2 = 1249	1	2
1249:2 = 624	1	4
624:2 = 312	0	8
312:2 = 156	0	16
156:2 = 78	0	32
78:2 = 39	0	64
39:2 = 19	1	128
19:2 = 9	1	256
9:2 = 4	1	512
4:2 = 2	0	1024
2:2 = 1	0	2048
1:2 = 0	1	4096

22/09

Data Representation

S & M → overflow not detected

```
+ 0 1 1 1   (+7) ← bigger magnitude
  1 0 1 1   (-3)
  ---------
  0 1 0 0   (+4)
```

1C → overflow not detected

```
    0 1 1 1   (7)
+   1 1 0 0   (-3) → 0011₂ = 3₁₀
  ---------
1 | 0 0 1 1
      1 +
  ---------
    0 1 0 0   (+4)
```

carry out

2C → overflow detected iff MSB of both is the same, the sum should have the same MSB (0→+ | 1→-)

```
    0 1 1 1   (7)
+   1 1 0 1   (-3) → 0011₂ = 3₁₀
  ---------
1 | 0 1 0 0
```

carry out

integer values 105949 ÷ 105989
↳ Excess -105949

4224 for use in LCD Clock
↳ BCD

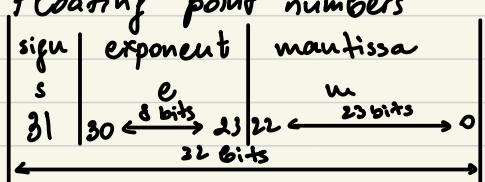
Fixed point fractions

x	2^{-x}
0	1
1	0.5
2	0.25
3	0.125
...	...

=> 8.75₁₀ ↓ 1000.11₂

e=0, value=0 | e=255 value=±∞, m=0

Floating point numbers



value = $(-1)^s \times 2^{e-127} \times m$

$m' = 1.m$

$1 \leq e \leq 254$ excess -127 [-126; 127]

value = 9.75
fixed: 1000.11

1.00011×2^3

s	e	m
0	10000010	00011 0000...

+ 130 − 127 = 3 ; m' = 1.00011

s	e	m
1	10000000	01101000...

− 128 − 127 = 1 ⇒ 2^1 ; 1.0101 ⇒ $10.101_2 = 2_{10} + 0.5_{10} + 0.125_{10} = 2.625_{10}$; $= -2.625_{10}$

	Sign		Exponent		Mantissa	
+	0	+	10000010	3	00100111000000000000000	→ 1001.00111
+	0	+	10000011	4	00101000110000000000000	→ 10010.100011
	0		10000011		10111110001	11011.110001

	s	exponent	mantissa	
+	0	10000011	00110110 1000...	010011.01101
	0	10000100	0010 1111 1000...	100101.1111
	0	10000100	1100101011000...	111001.01011

5

27/09

Instructure Set Architecture

stack based → CISC
register based → RISC

VLIW → very long instruction word
MISC → minimal instruction set computer

Memory Layout

bits -> flip-flop
words -> parallel access register
memory chip (SoC)
byte addressable
KB = KiB (binary kilobyte) = 1024 bytes

4 GB = $2^{10} \cdot 2^{10} \cdot 2^{10} \cdot 2^2$ -> 32 Gbytes -> 35 bits -> 30 words of 32 bits

big endian memory: ...|a|b|c|... -> can be processed as
little endian memory: ...|c|b|a|... -> waits for last one

32 registers -> 5 bit reg operand
3-reg instruction -> 15 bit => 32 - 15 = 17 bit OP
reg+dir in 16 MB memory
↳ 29 ↓ 2^{24}
↳ 32 - 29 = 3 bit OP => $2^3 = 8$ instructions
64 reg + reg + imm +? 16 bit const
↳ 2^6 ↳ 6 + 5 (reg) + 5 + 16 = 32 ✓

29/09

Instructure Set Architecture

register transfer notation
[R1] - register
M(A) - memory (direct)
TOS - top of stack

Accumulator = Memory + TOS (Processor)
Stack = TOS + Stack (Processor)
Register-Memory = Memory + registers (Processor)
Register - Register = registers (Processor)

1. ACCU-based ISA
   - single-register
   - simple design -> easy to build and program
   - hard access to data (bottleneck memory)
   - example: 16-bit words, byte-addressable
     - S & M integers
     - instruction -> 4 OP, 12 operand -> 4 KiB
     - IS: Data copy (load, store), Arithmetic & Logic, Flow control
     - LOAD 3 take M(3)
     - SUB 3 sub M(3) M(3)=0
     - STORE 3 store in M(3)

2. Register-based ISA
   - general purpose registers
     - faster than memory
     - fewer address bits
   - bottleneck memory
   - EA = effective address
   - index operand: EA = [Reg] + Idx
   - Scale s = 1, 2, 4 or 8
   - Displacement disp = 8, 16, 32 or 64 bit

3. Multiple-register Architecture

CISC style: 15 bytes at most

- Legacy prefixes (1-4 bytes, optional)
- Opcode with prefixes (1-4 bytes)
- ModR/M (1 byte, if required)
- SIB (1 byte, if required)
- Displacement (1/2/4/8 bytes, if required)
- Immediate (1/2/4/8 bytes, if required)

RISC (Reduced Instruction Set)

KISS (Keep it simple and stupid)

trade-off: CISC vs. RISC
HW vs. SW

Pro-CISC
easier to program
reduced code size
complex hard- not software
+legacy

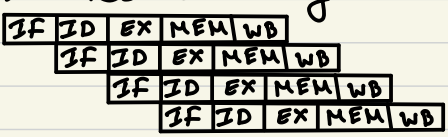
Con-CISC
complex encoding
maybe — slow
large energy consump

11/10

Basic Processing Unit (CPU)

RISC

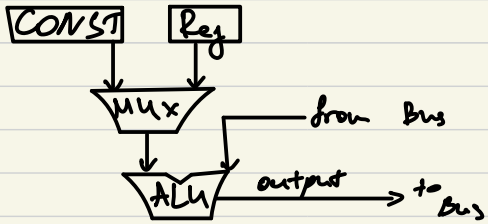
1. Fetch instr f/m mem
2. Decode instr and read reg
3. Execute ALU op
4. Memory access
5. Write back to reg

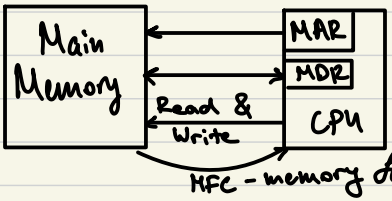


CISC

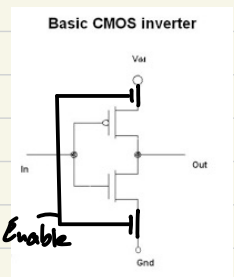
MAR – Memory Address Register (sends to memory, receives return)
MDR – Memory Data Register (send and receive from mem)
! Instruction address generator (PC)

2. Memory
3. MAR/MDR
4. IR (to control circuitry)

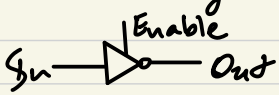




MFC - memory function complete



Tri-state gate



Enable	In	Out
0	X	Z
1	0	1
1	1	0

ADD R2, (R1)

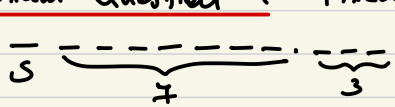
Exec

1. R_1 out, X_{IN}
2. R_1 out, MAR_{IN}, READ, WFMC (WMFC?) ← Wait For Memory to Complete
3. MDRout, ADD, Zin
4. ~~R_1 out, MAR_{IN}~~
5. Zout, MDR_{IN}, WRITE, WMFC

1. R_1 out, MAR_{IN}, READ
2. R_2 out, X_{IN}, WMFC
3. MDRout, ADD, Z_{IN}
4. Zout, MDR_{IN}, WRITE, WMFC

13/10 Tutorial

Exam Question 9 - Fixed vs Floating Point



= 11 bits Fixed point

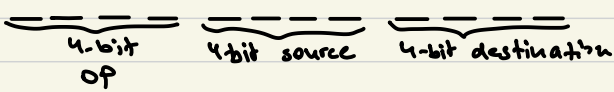
Largest positive - 126 2^7-2 1.111110 (mantissa) +6 exp.
smallest positive - 0.125 2^{-3} 0.001 +6 exp. 1 sign bit
31.5 $32-0.5$ $11111.1 \to 1.1111 \cdot 2^{+4}$ 5 bit mantissa
4 bit exponent
10 bits Floating point

Exam Question 11 - 16 bit ISA

$16\,MiB = 2^4\,MiB = 2^4 \cdot 2^{20}\,B = 2^{24}\,B \to$ 24 bits addressing

solution - 16 B memory $= 2^4$ B → 4-bit dir
16 instructions $= 2^4 \to$ 4-bit op
16 registers $= 2^4 \to$ 4-bit reg

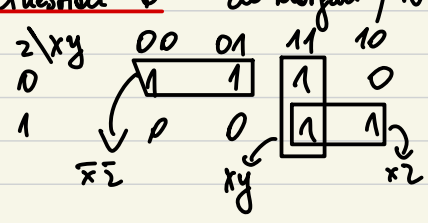
mov →



Question 8 - Bees

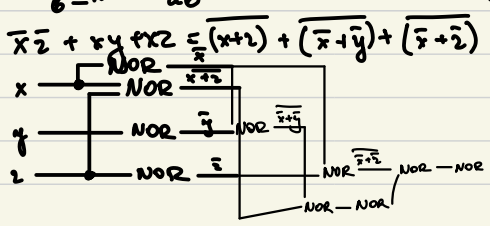
$65536 = 2^{16} \to$ overflow od 16 bits

Question 6 - de Morgan / NOR



$a = \overline{\bar{a}\bar{b}}$

$\bar{x}\bar{z} + xy + xz = \overline{(x+z)} + \overline{(\bar{x}+\bar{y})} + \overline{(\bar{x}+\bar{z})}$



bits for microcontrolling

2 (reg. add. +1) -> reg.

ex. 16 reg. -> 4 reg. add. -> 2(4+1) = 10

ALU -> # of op.

ex. 104 -> 7 bits

P - only JN - 1 bit

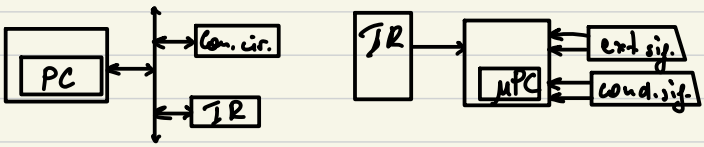
2 - JN and ours - 2 bits

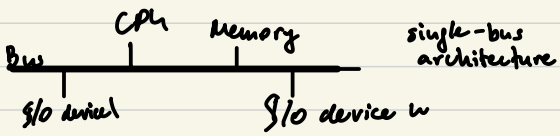
MAR - only JN - 1 bit

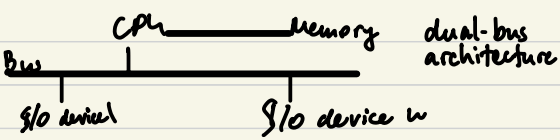
MDR - JN and ours - 2 bits

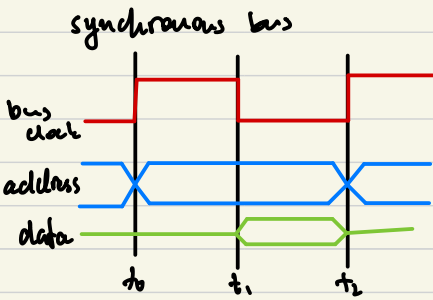
18/10

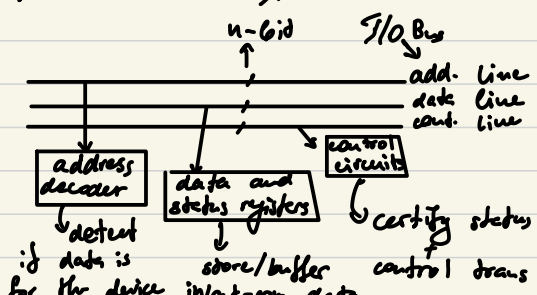
Input/Output



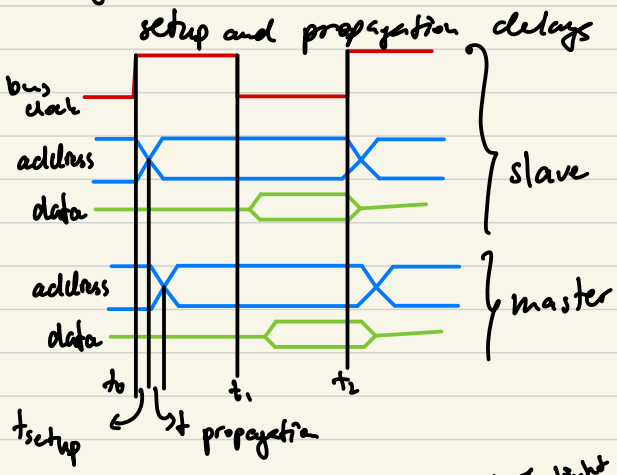








synchronous bus



bus driver delay 1.5ns

max bus length 15cm

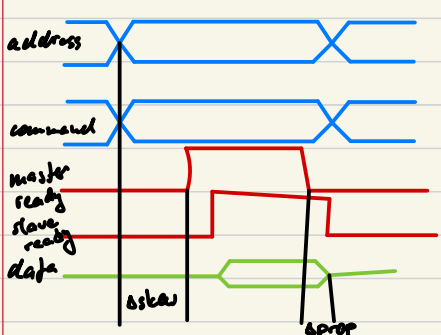
address decoder delay 6ns

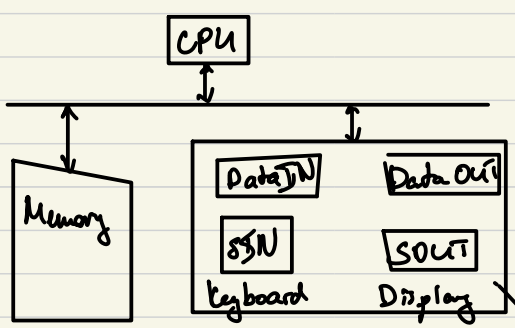
time to fetch data 3-15ns

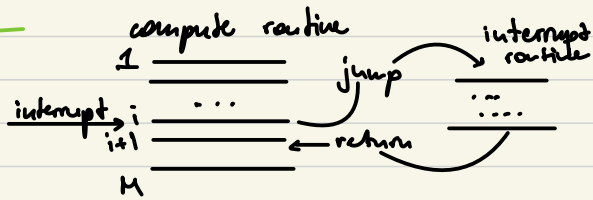
setup time (control circuitry) ≤2 ns

speed of light c in copper

$\frac{1.5 \times 10^{-2}\,m}{2 \times 10^{8}\,\frac{m}{s}} = 7.5 \times 10^{-11}\,s = 75\,ns$



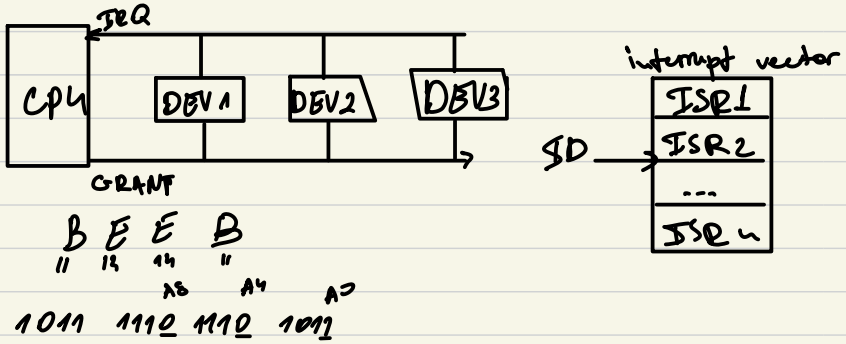




unconditional I/O

passive signaling (polling)

active signaling (interrupt)



memory-mapped I/O means I/O devices and memory use the same address space.

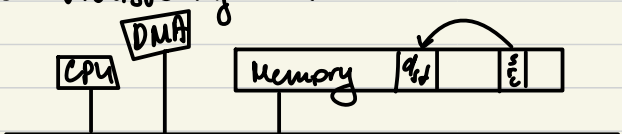
20/10

DMA & Memory Organization

DMA – direct memory address
1 instruction: MOVB (= COPY)

CPU		DMA
1. Issue transfer of block N+1 from disk to memory	→	1. receive command 2. transfer block N+1
2. start decoding block N	←	3. interrupt
3. handle interrupt		
4. finish decoding block N		

DMA – bus master when transferring data



32-bit RISC
8 registers
blockmove(src, dst, n)

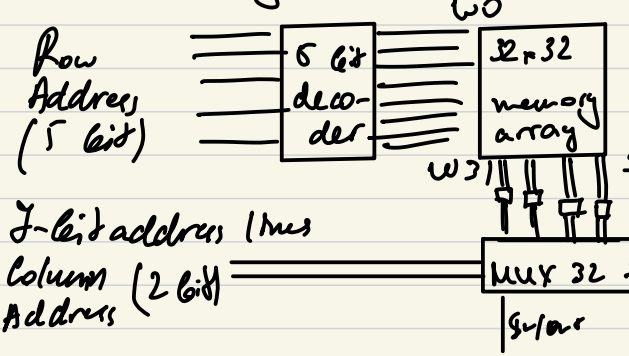
R_1 src
R_2 dst
R_3 n

```
Loop: MOV (R1), R4      ADD #4, R1
      MOV R4, (R2)      ADD #4, R2
      SUB #1, R3        BGZ loop RET
```

speed up = $\frac{\text{time(CPU)}}{\text{time(DMA)}} = \frac{6 \text{ c/w}}{2 \text{ c/w}} = 3$ → otherwise 2 cycle per word

$= \frac{6 \text{ c/w}}{3 \text{ c/w}} = 2$ → bus occupied every 3 cycles
⇒ 3 cycle per word for DMA

1024-bit memory - 128x8

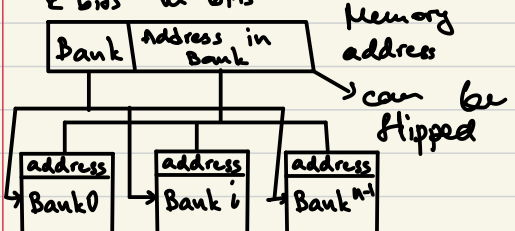


128x8	1024x1
7 address	10 address
8 data	1 data
2 R/W + CS	2 R/W + CS
2 power + gnd	2
19 pins	15 pins

32x16
5 address pins
16 data pins
2 R/W + CS
2 power + gnd
25 pins

GR = 16x ? [32]
one word = 32 bits
MDR = 32
MAR = $\log_2 32 + 1 = 6$

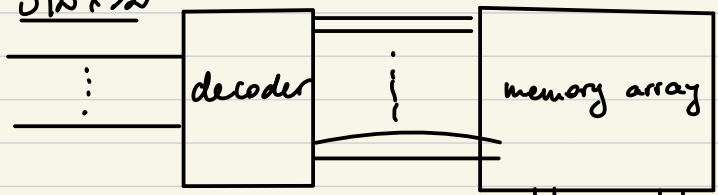
? x 32 memory organization
k bits n bits



20/10

Tutorial

512 x 32



2048 bytes = 2^{14} bits

5 column address line

$\frac{2^{14}}{2^5 \text{ data}} = 2^9$ address pin → 9 − 4 = 5 (row, column)

2^5 = 32 data pins (double word)

25/10

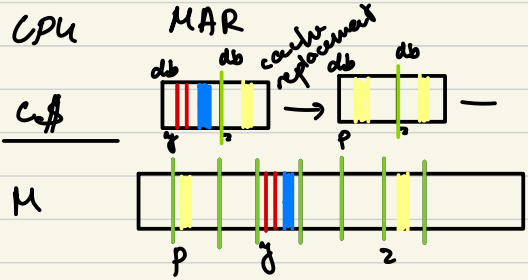
Caching

			Latency	Bandwidth
Primary (L1D, L1I) / Primary (L2)	CPU	compiler → Registers	1 cyc	< 1 KB
Secondary (L3)	CPU Chip, hardware	L1 Cache	1-3 cyc	32 KB - 1 MB
		L2 Cache	5-10 cyc	1-4 MB
Main Memory (L4)	Chips	DRAM	30-100 cyc	64 MB - 1 GB
Disk Secondary Memory	OS	Mechanical Disk	10^6-10^8 cyc	4 GB+

CPU ↓ cache ↓ memory

READ → check if data in cache → yes → HIT → read at address
↳ no → MISS → copy block into cache

WRITE → data in cache? yes → HIT
↳ MISS → write in main memory
— write in main memory (store through)
— set modified (dirty) bit and write later



- spatial → data close by locality
- temporal → used before data

L1 cache

cache hit ratio: $0 \le h \le 1$

access time ratio: c

cache miss ratio: $1-h$

access time main memory: m

$avg. = hc + (1-h)(c+m)$

$= c + (1-h)m$

$speedup = \frac{\text{time old}}{\text{time new}}$

$h = 0.85$

$m = 400\ ns$

$c = 40\ ns$

$s = \frac{m}{c + (1-h)m} = \frac{400}{40 + 0.15 \cdot 400} = 4$

$m = 400\ ns$

$h' = 0.925$

$c = 40\ ns$

$m' = 800\ ns$

$s' = \frac{m'}{c + (1-h')m} = \frac{800}{40 + 0.075 \cdot 400} = 8$

Cache Mapping

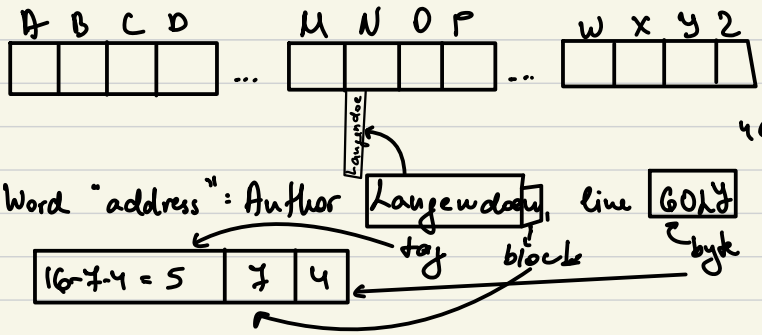
cache - smaller than memory (design)

main mem → library

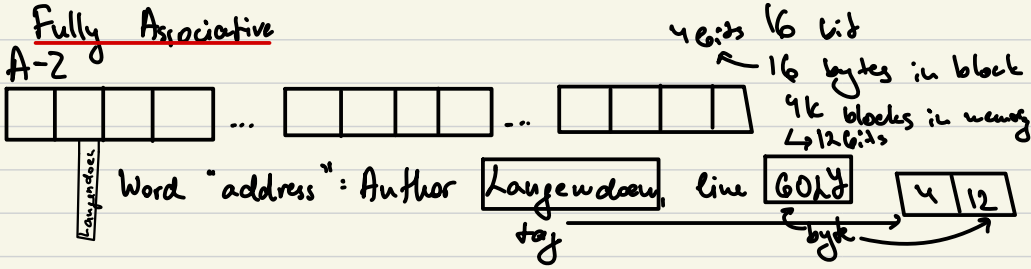
cache → bookcase at home (2 books at a time)

block → book

byte → line

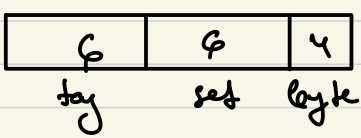


Fully Associative



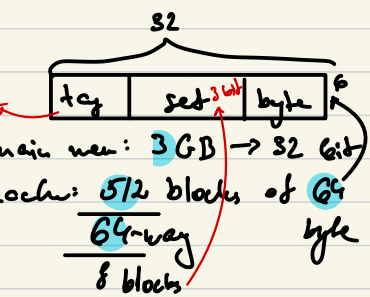
group block into sets
parallel search within set

2-way bis Associative

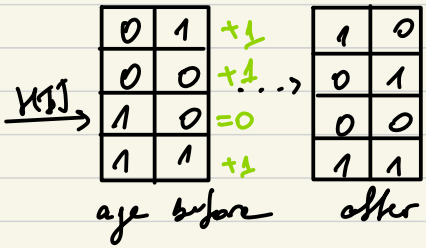


x-way associative
x=1 -> direct
x=|blocks in cache| -> fully associative

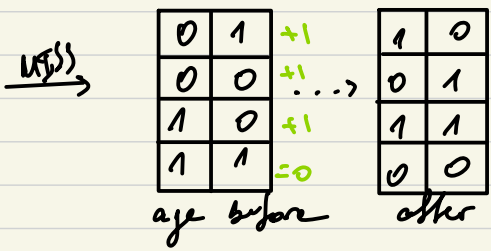
main memory: 9 GiB
cache: 512 blocks of 64 byte
↳ 9 bits tag ↳ 6 bits byte
32-way set-associative
↳ 5 bits set



Cache Replacement
2^x-way associative -> x-bit age counter per block

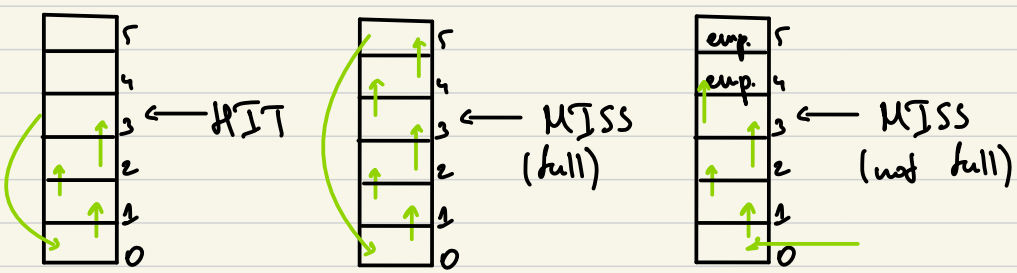


increased — Reset hit counter of hit block
increased — Increment lower counters
new at top
unchanged

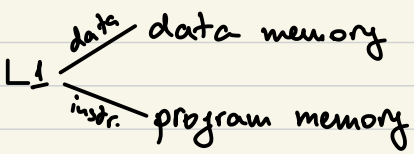


full:
increased — Replace block with highest counter, reset
increased — Increment all others
new at top

not full: — empty spot with 0 count
— Increment all others



Harvard Architecture:
instructions — spatial + temporal
data — temporal + spatial



27/10

Pipelining

1 washer -> 30 min
1 drier -> 50 min
1 folding mad -> 40 min
4 people

30	50	40	= 120 (one load)			
	30	50	40			
		30	50	40		
			30	50	40	

30 + 4 * 50 + 40 = 30 50 50 50 50 40 = 270

wash = 30 min
dry = 40 min
fold = 20 min

30	40	20	= 90 (one load)		
	30	40	20		
		30	40	20	
			30	40	20

30 + 4 * 40 + 20 = 210 = 30 40 40 40 40 20

Performance

Latency (L)
- time for one instruction to finish
- lower is better

Throughput (T)
- the # instructions per time unit
- higher is better
- time per instruction decreases (on average)

Increase clock frequency -> latency boost
problem: power wall

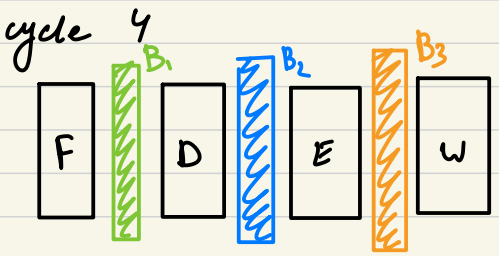
Multiple threads & cores -> throughput boost
problem: parallel programming

Fetch cpu Execute	Fetch 1	Fetch 2	Fetch 3	Fetch 4
	time →	Execute 1	Execute 2	Execute 3

Fetch instruction
Decode instruction and fetch operands
Execute instruction
Write result

clock cycle	1	2	3	4	5	6	7
I1	F1	D1	E1	W1			
I2		F2	D2	E2	W2		
I3			F3	D3	E3	W3	
I4				F4	D4	E4	W4

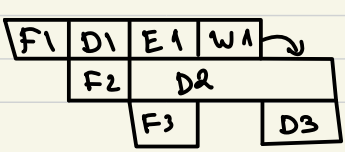
L = 4
T = 4



B_1 (F → D) I3
B_2 (D → E) I2
B_3 (E → W) I1

stalls:
- Cache miss [fetch, decode]
- Dependency between instructions [decode]
- Branching [exec]
- Long operation [exec]

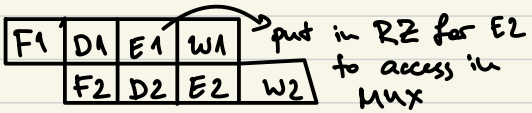
MUL R2, R3, R4 R2*R3=R4 I1
ADD R5, R4, R6 R5+R4=R6 I2



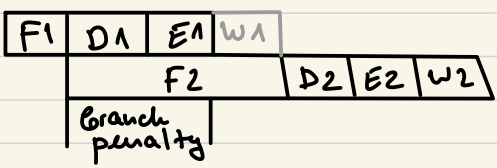
```
ADD R1, R2    R2+=R1
MUL R3, R4    R4+=R3
ADD R2, R3    R3*=R2

MOV R1, R2    R2=R1
ADD $100, R2  R2+=100
```

```
ADD R1, R2    R2+=R1
ADD R2, R3    R3+=R2
```




```
BGZ LOOP
MOV R3, SUM
```

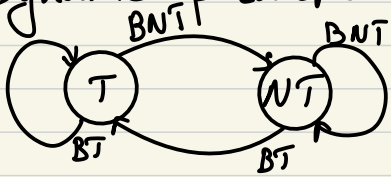



```
ADD R6, R7, R8          BGZ R3 LOOP
BGZ R3 LOOP      ⟶      ADD R6, R7, R8
MOV R8, SUM             MOV R8, SUM
```

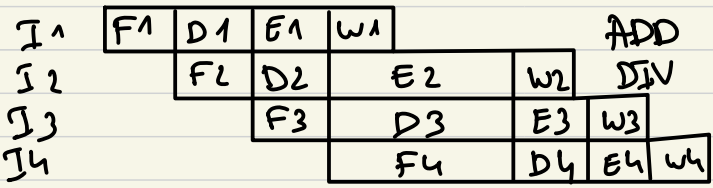
Static prediction

- backwards branches likely to be taken (loop)
- forward branches mostly skipped (rule vs. exception)
- sign bit (or compiler directed)

Dynamic prediction



ADD R2, R3
DIV R3, R1



Execution time of a program: T
Instruction count: N
cycles per instruction, CPI: S (e.g., 4)
Clock rate: R (e.g., 1 GHz)

without pipelining: $T = (N \times S)/R$
with n-stage pipeline: $T' = T/n$
but S increases due to stalls

$h_i = 0.95 \rightarrow miss = 0.05$
$h_d = 0.9 \rightarrow miss = 0.1$
cache miss → 32 cycles
20% memory access = 0.2
$0.35 * 32$

$m_i = 0.05$
$m_d = 0.1$
$p_{cache} = 32$
$f_{data} = 0.2$

$h_b = 0.8 \rightarrow miss = 0.2$
1 additional cycle
20% branching → 0.2
$0.2 * 0.2 * 1$

$m_{branch} = 0.2$
$p_{branch} = 1$
$f_{branch} = 0.2$

27/10 **Tutorial**

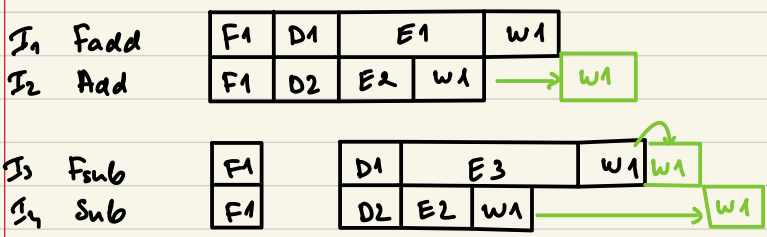
$h_{branch} = 0.90$ $m_{branch} = 0.10$
$p_{branch} = 1$ cycle
$f_{branch} = 0.20$

Cache misses
$= (m_{instr.} * f_{data} * m_{data}) * p_{cache}$
$= (0.05 + 0.3 * 0.1) * 16$
$= 1.28$

Branching
$= f_{branch} * m_{branch} * p_{branch}$
$= 0.2 * 0.1 * 1 = 0.02$

01/11

Superscalar Execution



Amdahl's Law

Parallelization can:
- → increase throughput
- → Latency remains same
- → reduce program runtime T

not everything can be parallelized
- → data dependencies
- → sequential algorithms

f_p → fraction of code that can be parallelized
$1 - f_p = f_s$ → sequencial execution

T_s → time to run program sequentially

$$T_p = T_s \times \left(f_s + \frac{f_p}{p}\right)$$

p → number of processors

$$S = \frac{1}{f_s + \frac{f_p}{p}}$$

$T_s = 10s$
$f_p = 80\%$ $f_s = 20\%$
$p = 4$

$T_p = 10\left(20\% + \frac{80\%}{4}\right) = 10.40\% = 4$

$S = \frac{T_s}{T_p} = \frac{10}{4} = 2.5$

1 core with 1 thread

$f_p = 20\%$ $f_s = 80\%$
$p = 8$

$S = \frac{T_s}{T_p} = \frac{T_s}{T_s\left(f_s + \frac{f_p}{p}\right)} = \frac{1}{80\% + \frac{20\%}{8}} = \frac{1}{82.5\%} = \frac{40}{33}$

S_{max} $p = \infty$ $= \frac{1}{f_s} = \frac{5}{4} = 1.25$

$s' = \frac{10}{9} = \frac{1}{f_s + \frac{f_p}{p}} = \frac{1}{0.8 + \frac{0.2}{p}}$ $\quad 0.8 + \frac{0.2}{p} = \frac{9}{10}$ $\quad p = 2$

$p = ?$ $\quad 0.8 + \frac{0.2}{2} = 0.9$ ✓

A certain program can be parallelized for 20% $\quad f_p = 20\% = 0.2 \quad f_s = 80\% = 0.8$

What is the speedup if we run the program on 8 processors? $\quad p = 8 \quad s = \frac{1}{f_s + f_p/p} = \frac{1}{0.8 + \frac{0.2}{8}} = \frac{40}{33}$

s = 40/33 . (If your answer is a decimal number, e.g., 0.75, enter the corresponding fraction, e.g., 3/4.)

What is the maximum speedup that can be obtained? $\quad p = \infty \quad s_{max} = \frac{1}{f_s} = 1.25$

s_{max} = 1.25 . (If your answer is a decimal number, e.g., 0.75, enter the corresponding fraction, e.g., 3/4.)

How many processors are required to get a speedup of $\frac{10}{9}$? $\quad p = ? \quad s' = \frac{1}{0.8 + \frac{0.2}{p}} = \frac{10}{9} \quad 0.9 = 0.8 + \frac{0.2}{p} \quad p = 2$

2 CPUs. (If your answer is a decimal number, e.g., 0.75, enter the corresponding fraction, e.g., 3/4.)

$f_p = 90\% \quad f_s = 10\%$

$p = 3 \quad s = \frac{1}{f_s + \frac{f_p}{p}} = \frac{1}{10\% + 30\%} = \frac{5}{2} = 2.5$

$s_{max} = \frac{1}{p=\infty} = \frac{1}{f_s + \frac{f_p}{p}} = \frac{1}{f_s} = 10 \quad s = 5 = \frac{1}{10\% + \frac{90\%}{p}} \quad 0.2 = 0.1 + \frac{0.9}{p}$

$p = ?$ $\quad p = 9$

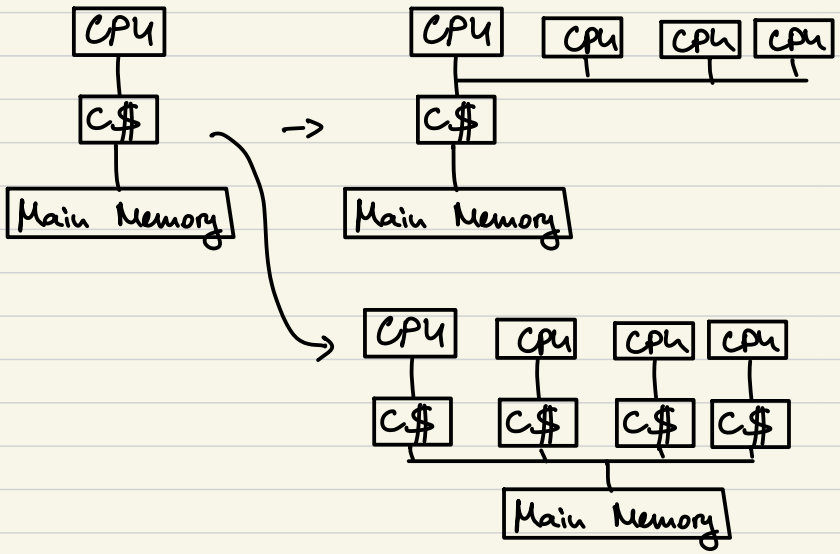
Flynn's Taxonomy

Single Instruction, Single Data (SISD)
→ conventional system

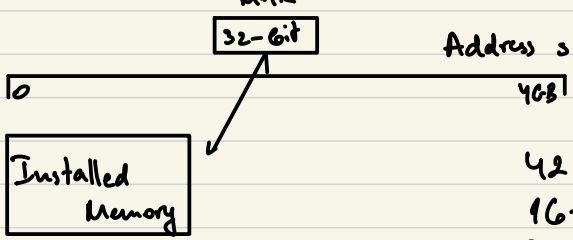
Single Instruction, Multiple Data (SIMD)
→ one instruction on multiple data streams

Multiple Instruction, Multiple Data (MIMD)
→ multiple instruction streams on multiple data streams

Multiple Instruction, Single Data (MISD)



Virtual Memory



64-bit computer $= 2^{64}$

16 KiB pages $= 2^{14}$

page table: $\frac{2^{64}}{2^{14}} = 2^{50}$

$2^{23} \cdot 4B = 2^{25}B =$ = 32 MiB

164-bit PC

8 byte word size

64 GiB word-addressable RAM

OS memory up to 128 GiB

pages of 8 KiB

42-bit PC

16-bit word size

8 GiB word-addressable RAM

OS memory up to 32 GiB

pages of 4 KiB

6 control bits

2^7 $\frac{128GiB}{8\,KiB} = \frac{2^{24}}{2^3}$ pages $\rightarrow$ 24 bits

offset: $4KiB = 2^{12}$ $\rightarrow$ 12 bits

$32\,GiB = 2^{35} \rightarrow$ 35 bits

$35 - 12 = 23$ bits

$8\,GiB = 2^{33}B$

33 bits − 12 bits offset

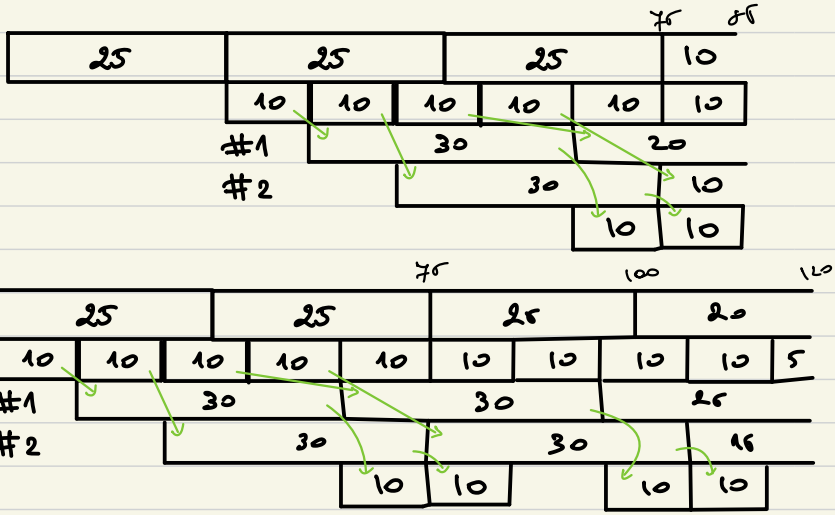
27 bits = 6 bits + 21 bits

32 bits = 4B

5 control bits

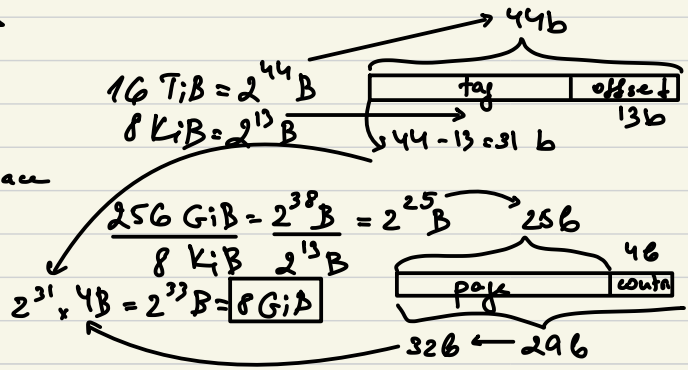
$\frac{2^{164}}{2^{24}} = 2^{140} \rightarrow 140 - 5 =$ = 135

$2^{135}B = 2^{115}$ MiB



Virtual Memory

64-bit processor
256 GiB memory
16 TiB address space
8 KiB pages
byte-aligned
4 control bits
page table size = ?



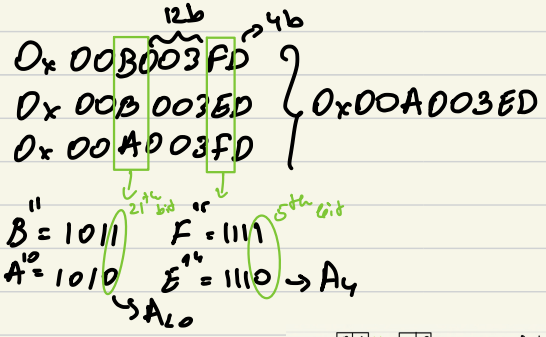
DP4

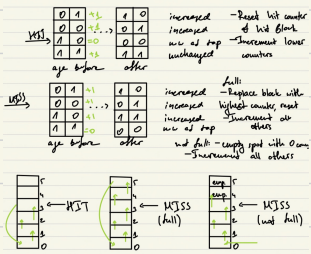
X, Cin → ALU inputs
memory access > 1 cycle
R2 → registers addressing
(R1) → indirect register addressing

a) R1 ← R2 + (R1)
1. R1out, MAR in, READ
2. R2 out, Y in, WMFC
3. MDRout, Xout, ALUadd, Zin
4. Zout, R1in, END

b) least amount of time
WMFC happens after last input reading

32 address lines
3 devices connected
4 ports
2 ignored address lines





Block 0 hit I
Block 3 hit III
Block 6 hit II
cache miss (full) IV

instructions

block	before	I	II	III	IV	after	
0	001	010	000	001	010		hit
1	010	011	011	100	101		
2	101	101	101	110	111		
3	110	110	110	000	001		hit
4	100	100	100	101	110		
5	011	000	001	010	011		hit
6	000	001	010	011	100		
7	111	111	111	111	000		cache miss