

1) Single precision floating number

[illegible]

(exponent - bias)

decimal representation  $(x) = (-1)^s \times (1 + \text{Fraction}) \times 2$

here, bias = 127.

$$S = 0.$$

$$E = (0111101)_2 = (125)_{10}$$

$$1 + \text{frac} = \text{significand} = \underbrace{1.0101 \dots 010}$$

$$(x) = (-1)^0 + (1.0101 \dots \overset{22 \text{ bits}}{010}) \times 2^{-2} = \underline{\underline{0.333333 \dots}}$$

2) ~~Decimal representation = 5.6617.~~

2) Decimal : 5.6677.

2) Decimal : 5.6677.  
Single precision floating point: 010000001011010101011011001100  
Decimal : 5.667699813843.

Value in Decimal : 5.667699813843.

difference: 1-861572265625E-7.

double : 01000000 00010110 10101011 10111001 10001100 11111100  
Go 101000 00100100.

Value in decimal : 5.667699999999999 G

distance :  $3.979039320257 E-17$ .

2) Decimal  $\rightarrow 5.6617$ .

In single precision floating point: 0 10000001 011010101011011000

This value in decimal: 5.66169999999999996

difference:

3) v.l.d vr1, [0x1000].

semantics:  $vr1 \leftarrow ([0x1000], [0x1004], [0x1008], [0x100C])$ .

Request.	Open	Closed.
(4, 15)	miss	miss
(4, 6)	conflict	miss
(6, 150)	conflict	miss
(7, 150)	hit	miss
(3, 9)	conflict	miss
(4, 9)	hit	miss
(4, 12)	conflict	miss
(4, 156)	conflict	miss
(5, 10)	conflict	miss
(6, 10)	hit	miss
(11, 10)	hit	miss

Open - row: Total misses = 7. Misses: 1; conflicts: 6.

Closed - row: Total misses = 11.



5)  $(4, 6), (3, 9), (4, 9), (5, 10), (6, 10), (11, 10), (4, 12), (4, 15),$   
 $(4, 150), (6, 150), (7, 150)$

6) It should be high with high spatial locality, <sup>in this case,</sup> number of hits will increase since, ~~same~~ application is likely to access the same row avoiding any ~~misses~~ conflicts.

7) There is no definite Yes/No.

→ Increases energy consumption: making predictions on biased branches.  
 predictions are not taken.

→ Decreases energy consumption: predictions are taken

8) weight vector =  $[10, -4, 19, -2, 2]$  → Bias.  
 BHR =  $[1, -1, 1, 1]$

$$\Rightarrow (10 \times 1) + (-4) + (19) + (-2) + 2 = 33.$$

since  $33 > 0$ .

⇒ Taken.

Given outcome is Taken so we update weight vector by as:

$$\text{weight vector} = [11, -5, 20, -1, 3].$$

$$\text{BHR} = [1, -1, 1, 1]$$

$$\Rightarrow 11 - 5 + 20 - 1 + 3 = 38 \Rightarrow \text{Taken}$$

9)  $\gamma$  increases soft error rate.

It increases soft error rate.  
Voltage scaling technique allows lower energy particles to flip bits unlike ~~earlier~~ previous case.

10) ld r1, 8[r2]  
st r1, 16[r3]

11) ~~sub r1, r2, r3~~  
~~add r2, r4, r5~~

WAR hazard: add r1, r2, r3.  
add r3, r2, r4.

```
12) [3] beq .b1.  
    [2] mov    r1, 0  
    [3] sub    r3, r4, r2.
```

~~13~~ [10] <sup>b1</sup>: add r4, r1, r3.

13) For the given sequence of instructions, there is load-use hazard. Hence, we must add bubble.

14)	IF	1	2	3					
	OF		1	2	3	3	3		
	EX			1	2	*	*	3	
	MA				1	2	*	*	3
	RW					1	2	*	*

\* implies pipeline bubble.



15)	Request	Time of arrival (ns)	Open-row	Closed-row
	$X_0$	0	40	40
	$Y_0$	10	100	100
	$X_1$	100	160	160
	$X_2$	200	220	240
	$Y_0$	250	310	360
	$X_3$	300	370	360

Row hit = 20 ns, Row-buffer conflict = ~~40~~<sup>60</sup> ns, Row-buffer miss = 40 ns.

16) strongly coupled MP vs

→ Programs run in parallel sharing code, data, files and network connections on a multiprocessor.

→ ~~own memory, which is not shared with other processes~~

→ usually share memory as they use same data.

loosely coupled MP.

→ Programs running are unrelated, run on parallel on a multiprocessor

→ ~~usually share mem~~

→ own memory, which is not shared with other processes.

17) Total chips in DRAM =  $\frac{8 \text{ Mb} \times 4}{1 \text{ Mb}} = \underline{\underline{32}}$

4 chips can be refreshed in parallel.

$\Rightarrow \frac{32}{4} = 8$  steps required to refresh all chips

Total time taken =  $8 \times \underbrace{(1000 \times 1000)}_{\text{Total number of cells}} \times 10^{-9} \times 100 \text{ s} = \underline{\underline{0.8 \text{ s}}}$

# 18) Simple Pipelining

001	fetch	001	decode	001	add	001	
002		002	fetch	002	decode	002	mul
003		003		003	fetch	003	add
004		004		004		004	decode
005		005		005		005	fetch
006		006		006		006	decode
007		007		007		007	fetch
008		008		008		008	decode
009		009		009		009	fetch
010		010		010		010	decode
011		011		011		011	fetch
012		012		012		012	decode
013		013		013		013	fetch
014		014		014		014	decode
015		015		015		015	fetch
016		016		016		016	decode
017		017		017		017	fetch
018		018		018		018	decode
019		019		019		019	fetch
020		020		020		020	decode
021		021		021		021	fetch
022		022		022		022	decode
023		023		023		023	fetch
024		024		024		024	decode
025		025		025		025	fetch
026		026		026		026	decode
027		027		027		027	fetch
028		028		028		028	decode
029		029		029		029	fetch
030		030		030		030	decode
031		031		031		031	fetch
032		032		032		032	decode
033		033		033		033	fetch
034		034		034		034	decode
035		035		035		035	fetch
036		036		036		036	decode
037		037		037		037	fetch
038		038		038		038	decode
039		039		039		039	fetch
040		040		040		040	decode
041		041		041		041	fetch
042		042		042		042	decode
043		043		043		043	fetch
044		044		044		044	decode
045		045		045		045	fetch
046		046		046		046	decode
047		047		047		047	fetch
048		048		048		048	decode
049		049		049		049	fetch
050		050		050		050	decode
051		051		051		051	fetch
052		052		052		052	decode
053		053		053		053	fetch
054		054		054		054	decode
055		055		055		055	fetch
056		056		056		056	decode
057		057		057		057	fetch
058		058		058		058	decode
059		059		059		059	fetch
060		060		060		060	decode
061		061		061		061	fetch
062		062		062		062	decode
063		063		063		063	fetch
064		064		064		064	decode
065		065		065		065	fetch
066		066		066		066	decode
067		067		067		067	fetch
068		068		068		068	decode
069		069		069		069	fetch
070		070		070		070	decode
071		071		071		071	fetch
072		072		072		072	decode
073		073		073		073	fetch
074		074		074		074	decode
075		075		075		075	fetch
076		076		076		076	decode
077		077		077		077	fetch
078		078		078		078	decode
079		079		079		079	fetch
080		080		080		080	decode
081		081		081		081	fetch
082		082		082		082	decode
083		083		083		083	fetch
084		084		084		084	decode
085		085		085		085	fetch
086		086		086		086	decode
087		087		087		087	fetch
088		088		088		088	decode
089		089		089		089	fetch
090		090		090		090	decode
091		091		091		091	fetch
092		092		092		092	decode
093		093		093		093	fetch
094		094		094		094	decode
095		095		095		095	fetch
096		096		096		096	decode
097		097		097		097	fetch
098		098		098		098	decode
099		099		099		099	fetch
100		100		100		100	decode
101		101		101		101	fetch
102		102		102		102	decode
103		103		103		103	fetch
104		104		104		104	decode
105		105		105		105	fetch
106		106		106		106	decode
107		107		107		107	fetch
108		108		108		108	decode
109		109		109		109	fetch
110		110		110		110	decode
111		111		111		111	fetch
112		112		112		112	decode
113		113		113		113	fetch
114		114		114		114	decode
115		115		115		115	fetch
116		116		116		116	decode
117		117		117		117	fetch
118		118		118		118	decode
119		119		119		119	fetch
120		120		120		120	decode
121		121		121		121	fetch
122		122		122		122	decode
123		123		123		123	fetch
124		124		124		124	decode
125		125		125		125	fetch
126		126		126		126	decode
127		127		127		127	fetch
128		128		128		128	decode
129		129		129		129	fetch
130		130		130		130	decode
131		131		131		131	fetch
132		132		132		132	decode
133		133		133		133	fetch
134		134		134		134	decode
135		135		135		135	fetch
136		136		136		136	decode
137		137		137		137	fetch
138		138		138		138	decode
139		139		139		139	fetch
140		140		140		140	decode
141		141		141		141	fetch
142		142		142		142	decode
143		143		143		143	fetch
144		144		144		144	decode
145		145		145		145	fetch
146		146		146		146	decode
147		147		147		147	fetch
148		148		148		148	decode
149		149		149		149	fetch
150		150		150		150	decode
151		151		151		151	fetch
152		152		152		152	decode
153		153		153		153	fetch
154		154		154		154	decode
155		155		155		155	fetch
156		156		156		156	decode
157		157		157		157	fetch
158		158		158		158	decode
159		159		159		159	fetch
160		160		160		160	decode
161		161		161		161	fetch
162		162		162		162	decode
163		163		163		163	fetch
164		164		164		164	decode
165		165		165		165	fetch
166		166		166		166	decode
167		167		167		167	fetch
168		168		168		168	decode
169		169		169		169	fetch
170		170		170		170	decode
171		171		171		171	fetch
172		172		172		172	decode
173		173		173		173	fetch
174		174		174		174	decode
175		175		175		175	fetch
176		176		176		176	decode
177		177		177		177	fetch
178		178		178		178	decode
179		179		179		179	fetch
180		180		180		180	decode
181		181		181		181	fetch
182		182		182		182	decode
183		183		183		183	fetch
184		184		184		184	decode
185		185		185		185	fetch
186		186		186		186	decode
187		187		187		187	fetch
188		188		188		188	decode
189		189		189		189	fetch



19) we should use ~~course~~ coarse grain multithreading strategies

20) v.cmp vr1, 50

v.gt.mul vr1, vr1, 50

21) Number of ranks per channel = 4  
64 bit wide data  $\Rightarrow$  8 chips per rank.

Total capacity of DRAM =  $8 \times 16 \text{Gb} \times 4$ . ; size of one chip = 16 Gb.  
= 64 Gb.

22) A C Z V K L Z V C M R B A K

Reuse distance = 6. [all access]

Reuse distance = 4 [distinct access]

23) ~~for (k=0; k < R; k++)~~

for (row = 0; row < R; row = row + B)

for (a = row; a < min(R, row + B); a++)

for (col = 0; col < C; col++)

for (t<sub>0</sub> = 0; t<sub>0</sub> < M; t<sub>0</sub>++)

for (t<sub>i</sub> = 0; t<sub>i</sub> < N; t<sub>i</sub>++)

for (i = 0; i < k; i++)

for (j = 0; j < k; j++)

{ Output - maps [t<sub>0</sub>] [row] [col] + = weights [t<sub>0</sub>] [t<sub>i</sub>] [i] \*  
Input - maps [t<sub>i</sub>] [s<sub>row</sub> + i] [s<sub>col</sub> + j]

24) a)  $\rightarrow N=1$  (or)  $N \geq 100$

say biased implies 99% taken or not not taken.

$\rightarrow$  Not conditions on array  $[ ]$ .

b)  $\rightarrow$  No condition on  $N$

$\rightarrow$  More than half of array should be divisible by 20

(or)  
More than half of array should be not divisible by 20.

25) Temporal multi-bit errors:

Use "scrubbing"  $\Rightarrow$  ~~read data~~ Uses ECC to correct data and then write that data.

$\rightarrow$  we need to correct data before it exceeds the capacity of ~~correction~~ corrigible errors.

Spatial multi-bit errors

Use "bit interleaving":

~~Changing each~~

Choose arrangement of blocks so that maximum ECC can be applied where each sub-block has error not exceeding capability of ECC.