CS2100

TUTORIAL #11

CACHE

(PREPARED BY: AARON TAN)

Q2. 4, 16, 32, 20, 80, 68, 76, 224, 36, 44, 16, 172, 20, 24, 36, 68.

MIPS: 1 word = 4 bytes
1 block = 2 words
16 words total = 8 blocks in total
2-way set associative
LRU replacement

Offset = 3 bits Set index = 2 bits

		Set index	Offset
4:	00…000	00	100
16:	00…000	10	000
32:	00…001	00	000
20:	00…000	10	100
80:	00…010	10	000
68:	00…010	00	100
76:	00…010	01	100
224:	00…111	00	000
36:	00…001	00	100
44:	00…001	01	100
16:	00…000	10	000
172:	00…101	01	100
20:	00…000	10	100
24:	00…000	11	000
36:	00…001	00	100
68:	00…010	00	100

Set	Valid	Tag	Word0	Word1	Valid	Tag	Word0	Word1
0	0				0			
1	0				0			
2	0				0			
3	0				0			

Q2. 4, 16, 32, 20, 80, 68, 76, 224, 36, 44, 16, 172, 20, 24, 36, 68.

MIPS: 1 word = 4 bytes
1 block = 2 words
16 words total = 8 blocks in total
2-way set associative
LRU replacement

Offset = 3 bits Set index = 2 bits

			Set index	Offset
→	4:	00…000	00	100
→	16:	00…000	10	000
→	32:	00…001	00	000
Hit →	20:	00…000	10	100
→	80:	00…010	10	000
→	68:	00…010	00	100
→	76:	00…010	01	100
→	224:	00…111	00	000
→	36:	00…001	00	100
→	44:	00…001	01	100
Hit →	16:	00…000	10	000
→	172:	00…101	01	100
Hit →	20:	00…000	10	100
→	24:	00…000	11	000
Hit →	36:	00…001	00	100
→	68:	00…010	00	100

Set	Valid	Tag	Word0	Word1	Valid	Tag	Word0	Word1
0	~~0~~ 1	~~0~~ ~~2~~ 1	~~M[0]~~ ~~M[64]~~ M[32]	~~M[4]~~ ~~M[68]~~ M[36]	~~0~~ 1	~~1~~ ~~7~~ 2	~~M[32]~~ ~~M[224]~~ M[64]	~~M[36]~~ ~~M[228]~~ M[68]
1	~~0~~ 1	~~2~~ 5	~~M[72]~~ M[168]	~~M[76]~~ M[172]	~~0~~ 1	1	M[40]	M[44]
2	~~0~~ 1	0	M[16]	M[20]	~~0~~ 1	2	M[80]	M[84]
3	~~0~~ 1	0	M[24]	M[28]	0			

Q3.

#	Code
i1	addi $s0, $zero, 0
i2	addi $s1, $s5, -1
i3	addi $s3, $zero, 1
i4	loop: slt $t0, $s0, $s1
i5	beq $t0 $zero, exit
i6	beq $s3, $zero, exit
i7	addi $t1, $s4, $s0
i8	lb $t4, 0($t3)
i9	addi $t3, $s4, $s1
i10	lb $t4, 0($t3)
i11	beq $t2, $t4, else
~~i12~~	~~addi $s3, $zero, 0~~
~~i13~~	~~j endW~~
i14	else: addi $s0, $s0, 1
i15	addi $s1, $s1, -1
i16	endW: j loop
i17	exit: [some instruction]

Tracing the first 10 iterations of the code
Assuming the string is a palindrome

First iteration:
i1 – i11, (skip i12 – i13), i14 – i16

Subsequent iterations:
i4 – i11, (skip i12 – i13), i14 – i16

Q3.

Direct mapped cache: 2 blocks, each 16 bytes

(a) Show instruction cache content at end of 1st iteration

(b) Calculate total cache hits after 10 iterations

First iteration: i1 – i11, i14 – i16

Subsequent iterations: i4 – i11, i14 – i16

Offset = 4 bits; Index = 1 bit

Addr. of i1: 0x4 = 00…000 0 0100 → index 0, word 1

Cache block 0	~~i0~~ ~~i8~~ i16	~~i1~~ ~~i9~~ i17	~~i2~~ ~~i10~~ i18	~~i3~~ ~~i11~~ i19
Cache block 1	~~i4~~ i12	~~i5~~ i13	~~i6~~ i14	~~i7~~ i15

First iteration: 9 hits

Each of next 9 iterations: 7 hits

Total hits = 9 + (7 × 9) = 72

First iteration:

inst	H/M?
i1	M
i2	H
i3	H
i4	M
i5	H
i6	H
i7	H
i8	M
i9	H
i10	H
i11	H
i14	M
i15	H
i16	M

Subsequent iterations

Q3.

Direct mapped cache: 4 blocks, each 8 bytes

(c) Show instruction cache content at end of 1st iteration

(d) Calculate total cache hits after 10 iterations

First iteration: i1 – i11, i14 – i16

Subsequent iteration: i4 – i11, i14 – i16

Cache block 0	~~i0~~ ~~i8~~ ~~i16~~ i8	~~i1~~ ~~i9~~ ~~i17~~ i9
Cache block 1	~~i2~~ i10	~~i3~~ i11
Cache block 2	i4	i5
Cache block 3	~~i6~~ ~~i14~~ ~~i6~~ i14	~~i7~~ ~~i15~~ ~~i7~~ i15

First iteration: 6 hits

Each of next 9 iterations: 7 hits

Total hits = 6 + (7 × 9) = 69

First iteration:

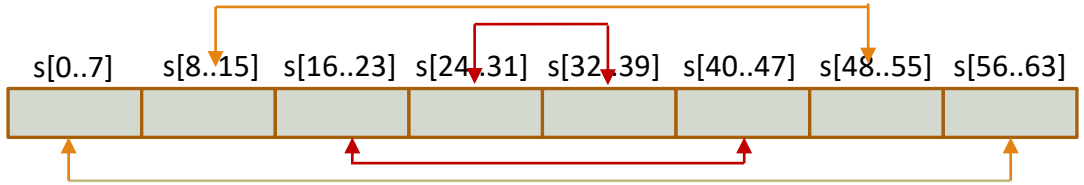
inst	H/M?
i1	M
i2	M
i3	H
i4	M
i5	H
i6	M
i7	H
i8	M
i9	H
i10	M
i11	H
i14	M
i15	H
i16	M

Subsequent iterations:

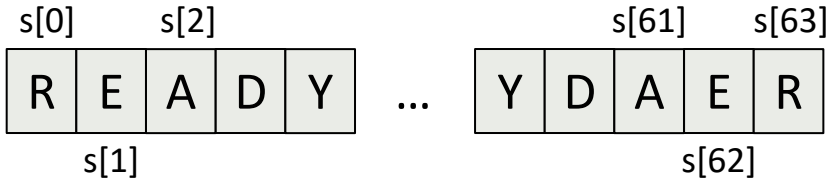
inst	H/M?
i4	H
i5	H
i6	M
i7	H
i8	M
i9	H
i10	H
i11	H
i14	M
i15	H
i16	M

Q3.

Direct mapped cache: 2 blocks, each 8 bytes



String is 64-character long and is a palindrome; first character at 0x1000



(e) Final content of data cache

(f) Hit rate

tag	index	offset
	1 bit	3 bits

Access pattern: s[0], s[63], s[1], s[62], …, s[31], s[32]

Cache block 0	~~**s[0..7]**~~	~~**s[48..55]**~~	~~**s[16..23]**~~	**s[32..39]**
Cache block 1	~~**s[56..63]**~~	~~**s[8..15]**~~	~~**s[40..47]**~~	**s[24..31]**

char	H/M?
s[0]	M
s[63]	M
s[1]	H
s[62]	H
:	
s[8]	M
s[55]	M
:	

Addr 0x1000 → block 0

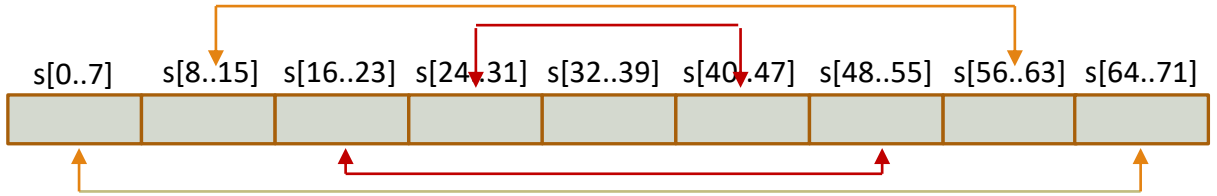
Addr 0x103F → block 1

For each block, first character is a miss, the remaining 7 characters accessed are hits.

Total % hits = 7/8= **87.5%**

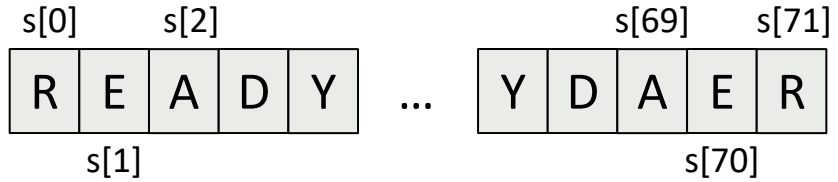
Q3.

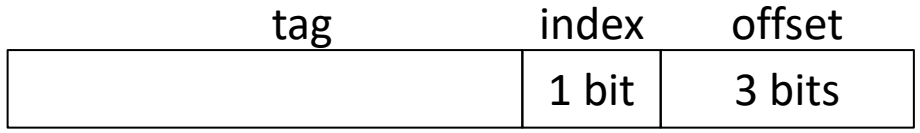
Direct mapped cache: 2 blocks, each 8 bytes



String is 72-character long and is a palindrome; first character at 0x1000

(g) Hit rate





Access pattern: s[0], s[71], s[1], s[70], …, s[35], s[36]

Cache block 0	~~s[0..7]~~ ~~s[64..71]~~ ~~s[0..7]~~ s[64..71]
Cache block 1	~~s[8..15]~~ ~~s[56..63]~~ ~~s[8..15]~~ s[56..63]

Data that go into cache block 0: s[0..7], s[64..71], s[16..23], s[48..55], s[32..39]

Data that go into cache block 1: s[8..15], s[56..63], s[24..31], s[40..47]

This is known as **cache thrashing**.

Only 7 hits in the last examined block s[32..39] → 7/72 = **9.72%**

char	H/M?	
s[0]	M	Addr 0x1000 → block 0
s[71]	M	Addr 0x1047 → **block 0!**
s[1]	M	
s[70]	M	
:		
s[8]	M	Addr 0x1008 → block 1
s[63]	M	Addr 0x103F → **block 1!**
s[9]	M	
s[62]	M	
:		

END OF FILE