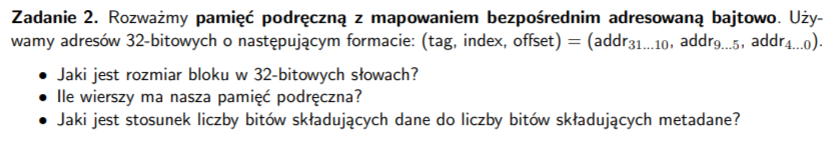


INDEX + ZNACZNIK + OFFSET = 12

Mamy 4 możliwe indeksy, wartości od 0 do 3, więc możemy to zawrzeć w 2 bitach. Znacznik jest wielkości 8 bitów, jak widać na przykładzie. Mamy w każdym bloku 4 bajty, więc potrzebujemy na oznaczenie tego 2 bitów, jak w przypadku indeksu. (TAG, INDEX, OFFSET) = (addr11..4, addr3..2, addr1..0)

|  |  |  |
| --- | --- | --- |
| Adres | Trafienie? | Wartość |
| 83 2 (10000011 00 10) | Tak | CC |
| 83 5 (10000011 01 01) | Nie | - |
| FF D (11111111 11 01) | Tak | C0 |

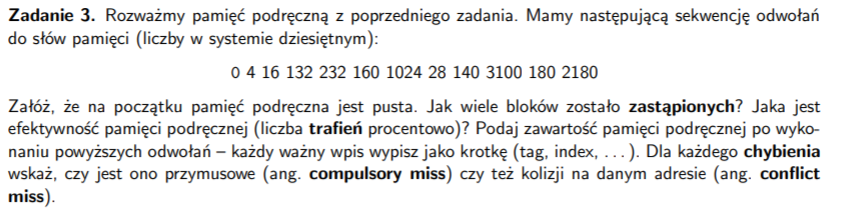


OFFSET (5) + INDEX (5) + TAG (22) = 32b

* Rozmiar bloku, czyli ile bajtów ma blok. Ze schematu adresu wynika, że jest to 32 (32 możliwości w offset).
* Mamy mapowanie bezpośrednie, więc jest tylko jedna linia na zbiór. Czyli ze schematu adresu wynika, że będzie to 32 (32 możliwości w index).
* Dane to faktyczne bajty, które przechowujemy w cache. Metadane to coś, co pomaga na obsługiwać cache. Czyli to będzie dla każdej linii:

|  |  |  |
| --- | --- | --- |
| Valid? | Tag | Dane |
| 1b | 22b | 32B\*8b=256b |

256b/23b = ~11,1304347826087



Rozważamy pamięć podręczną mapowaną bezpośrednio.

Adres jest postaci:

(TAG, INDEX, OFFSET) = (addr31..10, addr9..5, addr4..0)

DEC HEX BIN

0 = 0 = 00 00000 00000

4 = 4 = 00 00000 00100

16 = 10 = 00 00000 01010

132 = 84 = 00 00100 00100

232 = E8 = 00 00111 01000

160 = A0 = 00 00101 00000

1024 = 400 = 01 00000 00000

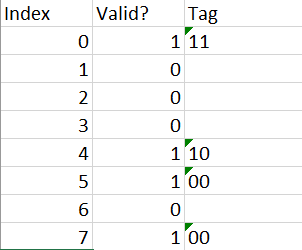
28 = 1C = 00 00000 11100

140 = 8C = 00 00100 01100

3100 = C1C = 11 00000 11100

180 = B4 = 00 00101 10100

2180 = 884 = 10 00100 00100



0 Compulsory

4 HIT

10 HIT

84 Compulsory

E8 Compulsory

A0 Compulsory

400 Conflict

1C Conflict

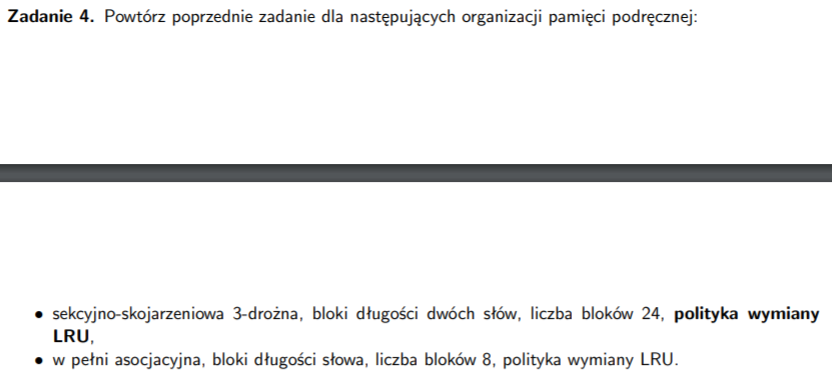
8C HIT

C1C Conflict

B4 HIT

884 Conflict

Trafienia (4/12) = 1/3 = ~33,(3)%



* Zakładam, że słowo to 64bity, czyli 8 bajty. W takim razie w tym przypadku blok będzie wielkości 16Bajtów. Polityka wymiany to LRU, czyli usuwam ten wpis, który był używany najwcześniej.

Pamięć jest sekcyjno-skojarzeniowa 3 drożna, więc będę miał trzy miejsca w danym indeksie.

Adres jest postaci:

(TAG, INDEX, OFFSET) = (addr31..7, addr6..4, addr3..0)

DEC HEX BIN

0 = 0 = 00000 000 0000

4 = 4 = 00000 000 0100

16 = 10 = 00000 000 1010

132 = 84 = 00001 000 0100

232 = E8 = 00001 110 1000

160 = A0 = 00001 010 0000

1024 = 400 = 01000 000 0000

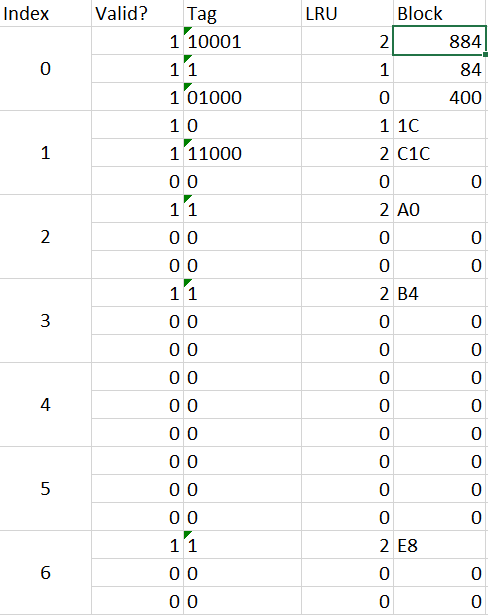
28 = 1C = 00000 001 1100

140 = 8C = 00001 000 1100

3100 = C1C = 11000 001 1100

180 = B4 = 00001 011 0100

2180 = 884 = 10001 000 0100



0 Compulsory

4 HIT

10 HIT

84 Compulsory

E8 Compulsory

A0 Compulsory

400 Compulsory

1C Compulsory

8C HIT

C1C Compulsory

B4 Compulsory

884 Conflict

3/12 = 25%

Jeśli słowo to 32bity:

* Zakładam, że słowo to 32bity, czyli 4 bajty. W takim razie w tym przypadku blok będzie wielkości 8Bajtów. Polityka wymiany to LRU, czyli usuwam ten wpis, który był używany najwcześniej.

Pamięć jest sekcyjno-skojarzeniowa 3 drożna, więc będę miał trzy miejsca w danym indeksie.

Adres jest postaci:

(TAG, INDEX, OFFSET) = (addr31..6, addr5..3, addr2..0)

DEC HEX BIN

0 = 0 = 000000 000 000

4 = 4 = 000000 000 100

16 = 10 = 000000 001 010

132 = 84 = 000010 000 100

232 = E8 = 000011 101 000

160 = A0 = 000010 100 000

1024 = 400 = 010000 000 000

28 = 1C = 000000 011 100

140 = 8C = 000010 001 100

3100 = C1C = 110000 011 100

180 = B4 = 000010 110 100

2180 = 884 = 100010 000 100

HEX

0 Compul

4 HIT

10 Compul

84 Compul

E8 Compul

A0 Compul

400 Compulsory

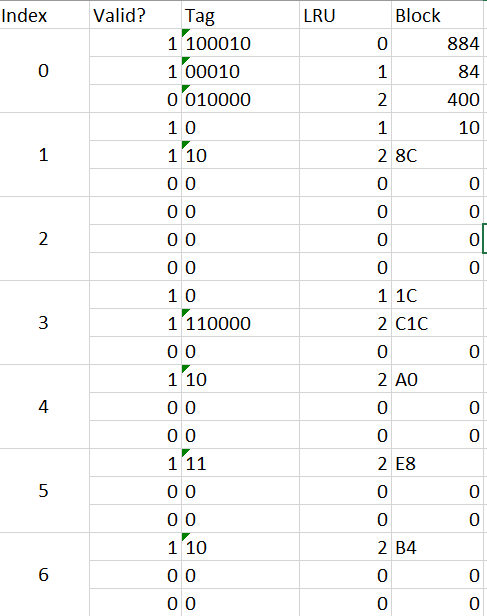
1C Compulsory

8C Compulsory

C1C Compulsory

B4 Compulsory

884 Compulsory



Trafienia: 1/12 = 8,(3)‬%

* Zakładam, że słowo to 64bity, czyli 8 bajty. W takim razie w tym przypadku blok będzie wielkości 8Bajtów. Polityka wymiany to LRU, czyli usuwam ten wpis, który był używany najwcześniej.

Pamięć jest w pełni asocjacyjna, więc mam wszystko w jednym. W tym przypadku w adresie mamy tylko tag i offset:

Adres jest postaci:

(TAG, OFFSET) = (addr31..7, addr3..0)

DEC HEX BIN

0 = 0 = 000000000 000

4 = 4 = 000000000 100

16 = 10 = 000000001 010

132 = 84 = 000010000 100

232 = E8 = 000011101 000

160 = A0 = 000010100 000

1024 = 400 = 010000000 000

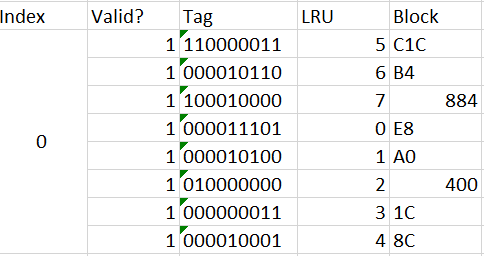
28 = 1C = 000000011 100

140 = 8C = 000010001 100

3100 = C1C = 110000011 100

180 = B4 = 000010110 100

2180 = 884 = 100010000 100



1/12 = 8,(3)%

* Zakładam, że słowo to 32bity, czyli 4 bajty. W takim razie w tym przypadku blok będzie wielkości 4Bajtów. Polityka wymiany to LRU, czyli usuwam ten wpis, który był używany najwcześniej.

Pamięć jest w pełni asocjacyjna, więc mam wszystko w jednym. W tym przypadku w adresie mamy tylko tag i offset:

Adres jest postaci:

(TAG, OFFSET) = (addr31..2, addr1..0)

DEC HEX BIN

0 = 0 = 0000000000 00

4 = 4 = 0000000001 00

16 = 10 = 0000000010 10

132 = 84 = 0000100001 00

232 = E8 = 0000111010 00

160 = A0 = 0000101000 00

1024 = 400 = 0100000000 00

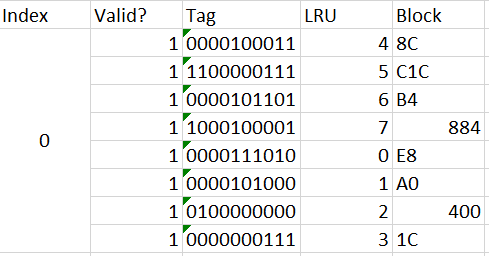
28 = 1C = 0000000111 00

140 = 8C = 0000100011 00

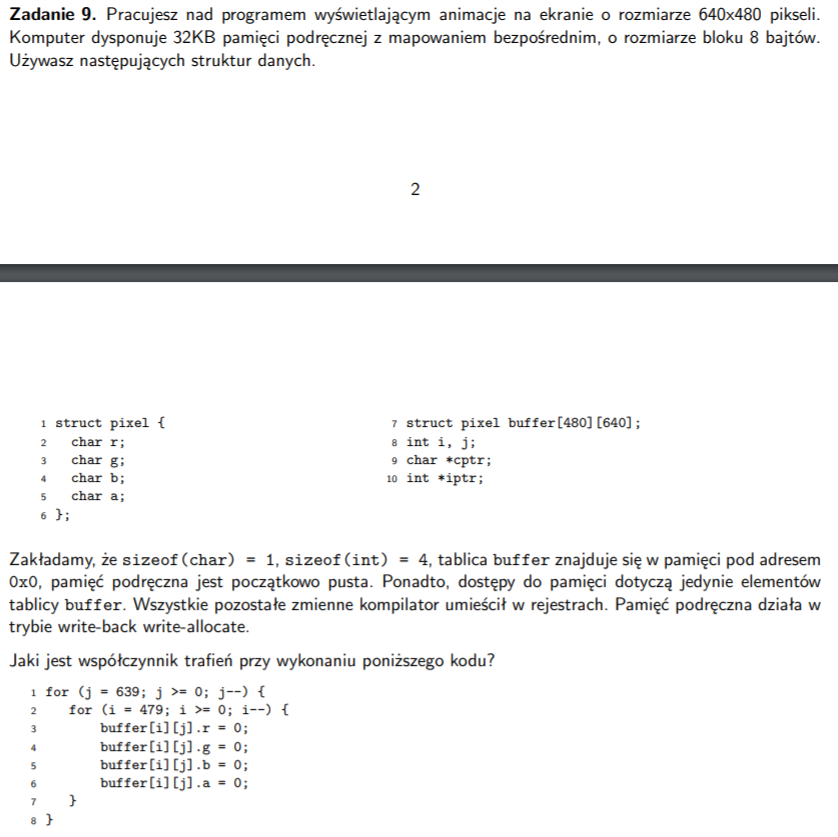
3100 = C1C = 1100000111 00

180 = B4 = 0000101101 00

2180 = 884 = 1000100001 00



0/12 = 0%



Mamy 32KB pamięci podręcznej mapowanej bezpośrednio. Rozmiar bloku to 8Bajtów.

32 768B = 8 (bajtów na blok) \* 4096 (linii)

Mamy ogólnie 307 200 dostępów do structów (tyle łącznie jest iteracji pętli wewnętrznej). Te dostępy mnożymy przez 4, bo dostajemy się do 4 pól tych structów: 1 228 800.

Dostępy to po kolei:

Buffer[479][639], buffer[478][639], Buffer[477][639], …, buffer[0][639]

Buffer[479][638], buffer[478][638], buffer[477][638], …, buffer[0][638]

…

Buffer[479][0], buffer[478][0], buffer[477][0], …, buffer[0][0]

Procesor do cache sprowadza dwa wiersze pionowo: Buffer[479][639] Buffer[479][638] potem buffer[478][639] buffer[478][638] itd. Wszystkich structów jest: 307 200‬ (po 4 bajty każdy). To daje 153 600 par tych structów (bo jak sprowadzamy to od razu dwa, żeby wypełnić blok). Ilość bloków cache: 4096. Tyle par structów może się zmieścić na raz do cache. Potem są zamieniane już wcześniej włożone dane.

Czyli licząc iteracje „miss” i „hit” mamy 480 missów. Potem w kolejnym przejściu pętli „j” mamy 480 hitów (bo sprowadziliśmy już te structy do pamięci). Schemat jest taki: 480 miss, 480 hit. A potem dochodzimy do momentu, gdzie cache się przepełnia. Dzieje się to:

480 + 480 + 480 + 480 + 480 + 480 + 480 + 480 = 3840 Par structów po 8 iteracji pętli zewnętrznej. Potem jest 9 iteracja pętli zewnętrznej i tutaj docieramy do 256 pary. To limit naszego cache. Kolejna iteracja będzie nadpisywać wcześniejsze pary w cache. Ale to jest nieparzysta iteracja, a w każdej nieparzystej iteracji my sprowadzamy do cache.

Kiedy będzie kolejny problem:

Jakby było:

buffer[3][3], buffer[2][3], buffer[1][3], buffer[0][3]

buffer[3][2], buffer[2][2], buffer[1][2], buffer[0][2]

buffer[3][1], buffer[2][1], buffer[1][1], buffer[0][1]

buffer[3][0], buffer[2][0], buffer[1][0], buffer[0][0]

Pierwszy wstawia do cache buffer[3][3] oraz buffer[3][2]. Kolejna iteracja też sprowadza do cache, tym razem buffer[2][3] oraz buffer[2][2]. Kolejna też i kolejna też. Potem mamy iterację: buffer[3][2]. No i to już mamy w cache, więc nie musimy sprowadzać. Potem buffer[2][2], też mamy. Tak samo [1][2] oraz [0][2]. [3][1] nie ma, sprowadzamy razem z [3][0]. Itd., więc licząc w sumie, co drugą iterację mamy hit. Za każdym hitem są 4 dostępy. Przy każdym miss jest tak: miss i 3 hity. Czyli dla mniejszego przypadku mamy 7/8. Nieźle.