Systemy komputerowe

Lista zadań nr 8

Karolina Szlęk 300411

Zadanie 1

Zadanie 1. Mamy system z pamięcią operacyjną adresowaną bajtowo. Szerokość szyny adresowej wynosi 12. Pamięć podręczna ma organizację sekcyjno-skojarzeniową o dwuelementowych zbiorach, a blok ma 4 bajty. Dla podanego niżej stanu pamięci podręcznej wyznacz, które bity adresu wyznaczają: offset, indeks, znacznik (ang. *tag*). Wszystkie wartości numeryczne podano w systemie szesnastkowym.

Indeks	Znacznik	Valid	B0	B1	B2	B3
0	00	1	40	41	42	43
	83	1	FE	97	CC	D0
1	00	1	44	45	46	47
	83	0	–	–	–	–
2	00	1	48	49	4A	4B
	40	0	–	–	–	–
3	FF	1	9A	C0	03	FF
	00	0	–	–	–	–

Określ, które z poniższych operacji odczytu wygenerują trafienie i ew. jakie wartości wczytają:

Adres	Trafienie?	Wartość
832	...	...
835	...	...
FFD	...	...

Adres	Trafienie	Wartość
832	TAK	CC
835	NIE	
FFD	TAK	C0

8 bitów na tag, 2 na indeks i 2 na offset

Dla podanego niżej stanu pamięci podręcznej wyznacz, które bity adresu wyznaczają: offset, indeks, znacznik (ang. *tag*).

→ Tag (znacznik) - pierwsze 8 bitów adresu
→ Indeks - kolejne dwa bity adresu (9-ty i 10-ty)
→ Offset - ostatnie dwa bity adresu

832(hex) = 1000 0011 0010(bin)

→ tag = 1000 0011 = 83
→ index = 00 = 0
→ offset = 10 = 2
0 -> 83 -> B2

835 = 1000 0011 0101

→ tag = 1000 0011 = 83
→ index = 01 = 1
→ offset = 01 = 1

Zadanie 2

Zadanie 2. Rozważmy **pamięć podręczną z mapowaniem bezpośrednim adresowaną bajtowo**. Używamy adresów 32-bitowych o następującym formacie: (tag, index, offset) = ($\text{addr}_{31...10}$, $\text{addr}_{9...5}$, $\text{addr}_{4...0}$).

- Jaki jest rozmiar bloku w 32-bitowych słowach?
- Ile wierszy ma nasza pamięć podręczna?
- Jaki jest stosunek liczby bitów składujących dane do liczby bitów składujących metadane?

Rozwiązanie:

1. Jaki jest rozmiar bloku w 32-bitowych słowach?
   Spójrzmy na krotkę tag, index, offset. Interesuje nas offset. (tag, index, offset) = (addr 31-10, addr9-5, addr4-0)Przeznaczone na niego są bity 0-4, łącznie 5 bitów, max = 11111, min=0 w ten sposób otrzymujemy 2^5 możliwości - 32 bajty.

2. Ile wierszy ma nasza pamięć podręczna?
   32 wiersze.
   Znowu spójrzmy na krotkę tag, index, offset. Interesować nas z niej będzie index. tag, index, offset) = (addr 31-10, addr9-5, addr4-0) Z polecenia widzimy, że indeks zajmuje bity 5-9,max =11111, min =0. Łącznie 5 bitów. Tak jak w podpunkcie pierwszym dostajemy 32 wiersze .

3. Jaki jest stosunek bitów składających dane do bitów składających metadane?

   Bity składające metadane, to bity tagu oraz bit valid.
   Bity składające dane, to bity bloku.

   Bity składające metadane - na tag przypadają bity 10 – 31. Są to łącznie 22 bity. Bit valid jest jeden. Dodajemy
   22+1=23
   Mamy 23 bity składające metadane.

   Bity składające dane. Wcześniej wyliczyliśmy, że blok ma 32 bajty. Chcemy wynik w bitach, żeby mówić o tych samych jednostkach.
   32*8 = 256 (bitów)
   Stosunek danych to metadanych: 256:23 (bity składające dane: bity składające metadane)

Mapowanie bezpośrednie (ang. direct mapping) - każdy blok w pamięci głównej jest odwzorowywany na tylko jeden możliwy wiersz pamięci (tzn. jeśli blok jest w cache'u to tylko w ściśle określonym miejscu)

Zadanie 3

Zadanie 3. Rozważmy pamięć podręczną z poprzedniego zadania. Mamy następującą sekwencję odwołań do słów pamięci (liczby w systemie dziesiętnym):

0 4 16 132 232 160 1024 28 140 3100 180 2180

Załóż, że na początku pamięć podręczna jest pusta. Jak wiele bloków zostało **zastąpionych**? Jaka jest efektywność pamięci podręcznej (liczba **trafień** procentowo)? Podaj zawartość pamięci podręcznej po wykonaniu powyższych odwołań – każdy ważny wpis wypisz jako krotkę (tag, index, . . .). Dla każdego **chybienia** wskaż, czy jest ono przymusowe (ang. **compulsory miss**) czy też kolizji na danym adresie (ang. **conflict miss**).

zastępywanie bloków - podmienianie wartości w pamięci cache potrzebnymi danymi
trafienie - gdy istnieje tag, index oraz valid = 1
Chybienie - eśli żądanej informacji nie ma w pamięci podręcznej, to mamy do czynienia z tzw. "chybieniem" i potrzebna informacja jest pobierana z pamięci operacyjnej z jednoczesnym wprowadzenie jej kopii do pamięci podręcznej.
compulsory miss - jeżeli nie ma nic: tagu, indexu (wczytujemy blok po raz pierwszy)
conflict miss - gdy nie ma twojego tagu pod danym indexem (nadpisanie pamięci)

Tak 22 bity, indeks 5, a offset 5

Mamy następującą sekwencję odwołań do słów pamięci (liczby w systemie dziesiętnym):

0 4 16 132 232 160 1024 28 140 3100 180 2180

Adres dziesiętny	(tag, index, offset)	hit or miss
0	0000 0000 0000	Compulsory miss
4	0000 0000 0100	Hit
16	0000 0001 0000	Hit
132	0000 1000 0100	Compulsory miss
232	0000 1110 1000	Compulsory miss
160	0000 1010 0000	Compulsory miss
1024	0100 0000 0000	Conflict miss, bo w cache'u 0 jest coś z tagiem 0
28	0000 0001 1100	Conflict miss, bo przed chwilą tag 0 zastąpiliśmy tagiem 1
140	0000 1000 1100	Hit - załadowaliśmy coś w zapytaniu z adresem 132
3100	1100 0001 1100	Conflict miss, jest tam tag 0
180	0000 1011 0100	Hit - załadowane w 160
2180	1000 1000 0100	Conflict miss, bo w 140 zaladowaliśmy tag 0

Pamięć podręczna po wykonaniu odwołań:

tag	Index	Offset
11	00000	11100
10	00100	00100
00	00101	10100
00	00111	01000

Efektywność: 4/12 = 1/3 = 33,(3)% <- ilość trafień/ ilość strzałów

Zadanie 4

Zadanie 4. Powtórz poprzednie zadanie dla następujących organizacji pamięci podręcznej:

- sekcyjno-skojarzeniowa 3-drożna, bloki długości dwóch słów, liczba bloków 24, **polityka wymiany LRU**,
- w pełni asocjacyjna, bloki długości słowa, liczba bloków 8, polityka wymiany LRU.

3-drożna - mamy 3 linie na jedną sekcję

Polityka LRU (ang. *Least recently used*) - polityka wybierania elementów do nadpisania w pamięci podręcznej polegająca na podmianie tych wartości, które były odczytywane najdawniej
Cyli jak nie mamy gdzie wstawić danych, bo wszystkie linie w secie są zajęte, to wywalamy ten, który najdawniej był używany

→ sekcyjno-skojarzeniowa 3-drożna, bloki długości dwóch słów, liczba bloków 24, **polityka wymiany LRU**,
E = 3 linie na indeks
B = 2, ponieważ bloki długości 2 słów
b = 1
S = 24 bloki / 3 linie na indeks = 8 zbiorów (indeksów)
s = 3
24 bloki, a bloki długości dwóch słów - w każdej linii blok ma 64

3 bity na indeks, 3na offset

Address	Result
0 (0000000 000 000)	Compulsory miss
4 (000000 000 100)	hit
16 (000000 010 000)	Compulsory miss
132 (000010 000 100)	Compulsory miss
232 (000011 101 000)	Compulsory miss
160 (000010 100 000)	Compulsory miss
1024 (010000 000 000)	Compulsory miss
28 (000000 011 100)	Compulsory miss
140 (000010 001 100)	Compulsory miss
3100 (110000 011 100)	Compulsory miss
180 (000010 110 100)	Compulsory miss
2180 (100010 000 100)	Conflict miss

Efektywność: 1/12 = 8,(3)%

→ w pełni asocjacyjna, bloki długości słowa, liczba bloków 8, polityka wymiany LRU.

→ S = 1
s = 0
E = 8
B = 1
b = 0
8 bloków, więc zapiszemy 8 wartości, każde kolejne będą nadpisaniem (w zadaniu mamy politykę wymiany LRU, więc aktualna wartość-każda kolejna po ósmej wartości, będzie

zastępowała najstarszą).
Znacznik (Tag) ma 7 bitów, indeks 0, a offset 5.

Adres	Trafienie?
0 (0000 0000 0000)	Compulsory miss
4 (0000 0000 0100)	Compulsory miss
16 (0000 0001 0000)	Compulsory miss
132 (0000 1000 0100)	Compulsory miss
232 (0000 1110 1000)	Compulsory miss
160 (0000 1010 0000)	Compulsory miss
1024 (0100 0000 0000)	Compulsory miss
28 (0000 0001 1100)	Compulsory miss
140 (0000 1000 1100)	Conflict miss
3100 (1100 0001 1100)	Conflict miss
180 (0000 1011 0100)	Conflict miss
2180 (1000 1000 0100)	Conflict miss

Efektywność: 0%

Zadanie 6

Zadanie 6. Wiemy, że im większa pamięć podręczna tym dłuższy czas dostępu do niej. Załóżmy, że dostęp do pamięci głównej trwa 70ns, a dostępy do pamięci stanowią 36% wszystkich instrukcji. Rozważmy system z pamięcią podręczną o następującej strukturze: L1 – 2 KiB, współczynnik chybień 8.0%, czas dostępu $0.66ns$ (1 cykl procesora); L2 – 1 MiB, współczynnik chybień 0.5%, czas dostępu $5.62ns$. Odpowiedz na pytania:

- Jaki jest średni czas dostępu do pamięci dla procesora tylko z cache L1, a jaki dla procesora z L1 i L2?
- Zakładając, że procesor charakteryzuje się współczynnikiem **CPI** (ang. *cycles per instruction*) równym 1.0 (bez wykonywania dostępów do pamięci), oblicz CPI dla procesora tylko z cache L1 i dla procesora z L1 i L2.

W architekturze komputerowej CPI cykle na instrukcję (inaczej cykle zegara na instrukcję, zegary na instrukcję lub CPI) to jeden aspekt wydajności procesora: średnia liczba cykli zegara na instrukcję dla programu lub fragmentu programu. Może zostać wrażone wzorem

time = (czas_dostępu do cache) + (współczynnik chybień) * (czas dostępu do pamięci głównej)

1. **Jaki jest średni czas dostępu do pamięci dla procesora tylko z cache L1?**

   Średni czas dostępu do pamięci dla procesora tylko z cache L1
   t1= 0,66ns+8,0%*70ns = 0,66ns+0,08*70ns=6,26ns

   Jaki jest średni czas dostępu do pamięci dla procesora z L1 i L2?

t2=0,66ns + 8,0%*(5.62ns+0,5%*70ns) =
= 0,66ns +0,08*(5.62ns+0.005*70ns) =
= 1,1376ns ≈ 1,1ns

2. **CPI** (ang.cycles per instruction) równe 1.0 (bez wykonywania dostępów do pamięci).
   CPI = 1 + time * ((procent dostępów do pamięci) / (czas dostępu do cache))
   dzielimy przez czas dostępu do cashe (0,66ns), żeby uzgodnić jednostki

0,66ns = 1C

CPI dla procesora tylko z cache L1
CPI=1+t1·36%/0,66ns
t1 – wyliczone w podpunkcie 1
t1= 0,66ns+8,0%*70ns = 0,66ns+0,8*70ns=6,26ns
CPI=1+ 6,26ns *36%/0,66ns = 1+ 6,26ns * 0,36/0,66ns = 4,414545≈4,41C

CPI dla procesora z L1 i L2
CPI=1+t2·36%/0,66ns
t2 – wyliczone w podpunkcie 1
t2=0,66ns + 8,0%*5.62ns+0,5%*70ns=
= 0,66ns +0,08*5.62ns+0.005*70ns =
= 1,1376ns ≈ 1,1ns
CPI=1+t2·36%/0,66ns = 1+1,1376ns·0,36/0,66ns = 1,620509 ≈ 1,62C

Zadanie 8

Zadanie 8. Dany jest następujący kod oraz założenia co do sposobu jego wykonania.

```
1 int x[2][128];
2 int i;
3 int sum = 0;
4
5 for (i = 0; i < 128; i++) {
6   sum += x[0][i] * x[1][i];
7 }
```

- `sizeof(int) = 4`
- Tablica x znajduje się w pamięci pod adresem 0x0
- Pamięć podręczna jest początkowo pusta
- Dostępy do pamięci dotyczą jedynie elementów tablicy x. Wszystkie pozostałe zmienne kompilator umieścił w rejestrach. Pamięć podręczna nie przechowuje instrukcji.

Oblicz współczynniki chybień przy wykonaniu tego kodu dla każdego z poniższych wariantów

- pamięć podręczna ma 512 bajtów, mapowanie bezpośrednie, rozmiar bloku 16 bajtów
- jak powyżej, z tym że rozmiar pamięci podręcznej to 1024 bajtów
- pamięć podręczna ma 512 bajtów, jest dwudrożna, sekcyjno-skojarzeniowa, używa polityki zastępowania LRU, rozmiar bloku to 16 bajtów

W tym ostatnim przypadku, czy zwiększenie rozmiaru pamięci podręcznej pomoże zredukować współczynnik chybień? A zwiększenie rozmiaru bloku?

Do rozwiązania przyda nam sie następujący wzór

C (cache size) = S * E * B

Gdzie:

S - number of sets

E - lines per set

B - block size

sizeof(int) = 4

1. pamięć podręczna ma 512 bajtów, mapowanie bezpośrednie, rozmiar bloku 16 bajtów

C: 512 bytes,
E: 1,
B: 16 bytes =2^4
S: $\log_2 32$

5 bloków

adres x[0][i] to 4 * i
adres x[1][i] to 512 + 4 * i

Zobaczmy co będzie się działo dla początkowych wywołań:

Dla i=4

x[0][4] = 00 0001 0000 (16 w systemie dziesiętnym) dostaniemy compulsowy miss
x[1][4] = 10 0001 0000 (528 w systemie dziesiętnym) dostaniemy conflict miss

Dla i=1

x[0][1] = 00 0000 0100 (4 w systemie dziesiętnym) dostaniemy conflict miss jest tam już x[1][0] wczytane zostaną elem w postaci 00 0000 ABCD
x[1][1] = 10 0001 0100 (516 w systemie dziesiętnym) dostaniemy conflict miss, wczytanie zostaną elementy w postaci 10 0000 ABCD

Dla i=0

x[0][0] = 00 0000 0000 (0 w systemie dziesiętnym) dostaniemy compulsowy miss, do pustego indeksu 0, wczytane zostaną elementy (00 0000 0000, 00 0000 0100, 00 0000 1000, 00 0000 1100) - postać elementów 00 0000 ABCD
x[1][0] = 10 0000 0100 (512 w systemie dziesiętnym) dostaniemy conflict miss, (w indeksie o zapisane już jest x[0][0], ale x[1][0] ma inny tag niż x[0][0].

Zmienna	Address	Tag	Index	Offset	Result
x[0][0]	0	0	00000	0000	Compulsory miss
x[1][0]	512	1	00000	0000	Conflict miss, nadpisze
x[0][1]	4	0	00000	0100	Conflict miss, nadpisze
x[1][1]	516	1	00000	0100	Conflict miss

Nie mieliśmy żadnych trafień
miss rate (współczynnik chybień): 100%

2. jak powyżej, z tym że rozmiar pamięci podręcznej to 1024 bajtów

C: 1024 bytes,

E: 1,

B: 16 bytes= 2^4

S: $\log_2 64$

Znowu zobaczmy co będzie się działo dla początkowych wywołań:

Dla i=4

x[0][4] = 00 0001 0000 (16 w systemie dziesiętnym) dostaniemy compulsory miss
przekazuje na indeks wyżej, bo nie zmieści się na 4 bitach.
Do indeksu 1 elem w postaci 00 0001 ABCD
x[1][4] = 10 0001 0000 (528 w systemie dziesiętnym) dostaniemy compulsory miss nie mieści się na 4 bitach. Wczytanie do indeksu 10 0001 elem w postaci 01 0001 ABCD

Dla i=1

x[0][1] = 00 0000 0100 (4 w systemie dziesiętnym) trafienie
x[1][1] = 10 0001 0100 (516 w systemie dziesiętnym) trafienie

Dla i=0

x[0][0] = 00 0000 0000 (0 w systemie dziesiętnym) dostaniemy compulsowy miss, do indeksu 0, wczytane zostaną elementy w postać 00 0000 ABCD
x[1][0] = 10 0000 0100 (512 w systemie dziesiętnym) dostaniemy compulsory miss. Wczytanie do indeksu 10 0000 elem w postaci 10 0000 ABCD

Zmienna	Address	Index	Offset	Result
x[0][0]	0	000000	0000	Compulsory miss
x[1][0]	512	100000	0000	Compulsory miss
x[0][1]	4	000000	0100	hit
x[1][1]	516	100000	0100	hit
x[0][4]	16	000001	0000	Compulsory miss
x[1][4]	528	100001	0000	Compulsory miss

Jak podwoimy rozmiar pamięci podręcznej to wtedy cała tablica zmieści się w niej zmieści. Więc jedyne chybienia to te przymusowe chybienia dla każdego nowego zbioru. Dla i=0 mieliśmy miss, potem dla i=1,2,3 były trafienia, a dla i=4 znowu nie.
miss rate (współczynnik chybień): ¼ = 25%

3. pamięć podręczna ma 512 bajtów, jest dwudrożna, sekcyjno-skojarzeniowa, używa polityki zastępowania LRU, rozmiar bloku to 16 bajtów

C: 512 bytes,

E: 2,

B: 16 bytes,

S: $\log_2 16$

Znowu zobaczmy co będzie się działo dla początkowych wywołań:

Dla i=4

x[0][4] = 00 0001 0000 (16 w systemie dziesiętnym) dostaniemy compulsowy miss
przekazuje na indeks wyżej, bo nie zmieści się na 4 bitach.
Do indeksu 1 elem w postaci 00 0001 ABCD (blok1)
x[1][4] = 10 0001 0000 (528 w systemie dziesiętnym) dostaniemy compulsory miss przekazuje na indeks wyżej, bo nie zmieści się na 4 bitach. Wczytanie do 10 0001 elem w postaci 01 0001 ABCD(blok2)

Dla i=1

x[0][1] = 00 0000 0100 (4 w systemie dziesiętnym) trafienie
x[1][1] = 10 0001 0100 (516 w systemie dziesiętnym) trafienie

Dla i=0

x[0][0] = 00 0000 0000 (0 w systemie dziesiętnym) dostaniemy compulsory miss, do indeksu 0, wczytane zostaną elementy w postać 00 0000 ABCD (blok1)
x[1][0] = 10 0000 0100 (512 w systemie dziesiętnym) dostaniemy compulsory miss. Wczytanie do indeksu 0 elem w postaci 10 0000 ABCD (blok2)

Zmienna	Address	Tag	Index	Offset	Result
x[0][0]	0	00	0000	0000	Compulsory miss
x[1][0]	512	10	0000	0000	Compulsory miss
x[0][1]	4	00	0000	0100	hit
x[1][1]	516	10	0000	0100	hit
x[0][4]	16	00	0001	0000	Compulsory miss
x[1][4]	524	10	0001	0000	Compulsory miss

W pierwszym podpunkcie zapisywaliśmy wartości w tej samej linii. Teraz jeśli adres będzie ten sam a tagi będą się różnić to możemy je zapisać w tym samym zbiorze. Unikamy dzięki temu conflict missów. Rozmiar bloku to 16 bajtów, więc za każdym razem wczytujemy maksymalnie 4 inty. Więc compulsory miss będzie występować co czwartą wartość.

miss rate (współczynnik chybień): ¼ = 25%

W tym ostatnim przypadku czy zwiększenie rozmiaru pamięci podręcznej pomoże zredukować współczynnik chybień?

Nie, zwiększenie rozmiaru pamięci podręcznej nie poprawi współczynnika chybień, bo naszym problemem są już tylko compulsory miss. Współczynnik chybień, który dostajemy na chwilę obecną jest wystarczająco dobry.

A zwiększenie rozmiaru bloku?

Zwiększenie rozmiaru bloku spowoduje, że będziemy mogli wczytać więcej int w tym samym czasie. Zatem pomoże też zredukować współczynnik chybień.

Zadanie 10

Zadanie 10. Kod z poprzedniego zadania został zoptymalizowany przez dwóch programistów. Wynik ich pracy znajduje się odpowiednio w lewej i prawej kolumnie poniżej.

```
1 char *cptr = (char *) buffer;
2 while (cptr < (((char *) buffer) + 640 * 480 * 4)) {
3       *cptr = 0;
4        cptr++;
5 }
```

```
1 int *iptr = (int *)buffer;
2 while (iptr < ((int *)buffer + 640*480)) {
3       *iptr = 0;
4        iptr++;
5 }
```

Czy któryś z tych programów jest wyraźnie lepszy, jeśli idzie o wykorzystanie pamięci podręcznej? Odpowiedź uzasadnij wyliczając odpowiednie współczynniki trafień.

Na pierwszy rzut oka, wydaje się, że kod pirewszy (ten po lewej) będzie miał więcej iteracji, przez co potencjalnie może się dłużej wykonywać (choć niekoniecznie).
Przyjrzyjmy się dokładniej obu kodom.

Kod pierwszego programisty:

Idziemy co bajt, ponieważ jest char,

Jest jeden miss, pozostałe to trafienia (blok ma 8 bajtów)

Współczynnik trafień to 7/8 (trafienia/próby) = 87,5%

Kod drugiego programisty:

Idziemy co bajt (jest int),

Blok ma 8 bajtów - jeden miss i jedno trafienie (razem 2 “próby” daje 8 bajtów, więcej nie możemy w bloku)

Współczynnik trafień to 1/2 (trafienia/próby) = 50%

Dla obu kodów mamy miss przy wczytaniu bloku, więc pod tym względem się nie różnią.

Sprawdzić odniesienia do RAM.

Kod pierwszy	Kod drugi
dostępy: 680 * 480 * 4 = 1228800 iteracje	dostępy: 640 * 480 = 307200
miss: 1/8 * 1228899 = 153600	Miss:½ * 307200 = 153600

W kodzie pierwszym do pamięci RAM odwołujemy się 1228800/8 razy, to znaczy 680 * 480 * 4/8 razy.
W kodzie drugim do pamięci RAM odwołujemy się 307200/2 razy, to znaczy 680 * 480/2 razy.

To znaczy, że do pamięci RAM odwołujemy się taką samą ilość razy. Jest tak samo dla obu kodów.

Zatem patrzymy na odwołania do cache. W drugim kodzie mamy mniej takich odwołań.

Można więc stwierdzić, że kod drugi ma mniej iteracji wiec będzie bardziej wydajny. Na samym początku, po wyliczeniu współczynnika trafień, można było sądzić, że jest na odwrót. Kod drugi miał większy miss rate, ale ze względy na mniejszą liczbę odwołań do pamięci możemy stwierdzić, ze będzie lepszy.