Systemy komputerowe

Lista zadań nr 8

Na zajęcia 4 – 7 maja 2020

UWAGA! W trakcie prezentacji rozwiązań należy być przygotowanym do wyjaśnienia pojęć, które zostały oznaczone **wytłuszczoną** czcionką.

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	...	...

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?

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**).

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.

Zadanie 5. Rozważamy system z dwupoziomową pamięcią podręczną z polityką zapisu *write-back* z *write-allocate*. Dodatkowo zakładamy, że blok o określonym adresie może znajdować się tylko na jednym poziomie pamięci podręcznej (ang. *exclusive caches*). Przy pomocy drzewa decyzyjnego przedstaw jakie kroki należy wykonać, by obsłużyć chybienie przy zapisie do `L1`? Nie zapomnij o bicie `dirty` i o tym, że pamięć podręczna może być całkowicie wypełniona! Zakładamy, że pamięć podręczna pierwszego poziomu nie może komunikować się bezpośrednio z pamięcią operacyjną.

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.

Zadanie 7. Dla czterodrożnej sekcyjno-skojarzeniowej pamięci podręcznej chcemy zaimplementować politykę wymiany LRU. Masz do dyspozycji dodatkowe $log_2(4!)$ bitów na zbiór. Nie można modyfikować zawartości linii w zbiorze, ani zamieniać elementów kolejnością. Jak wyznaczyć kandydata do usunięcia ze zbioru? Jak aktualizować informacje zawarte w dodatkowych bitach przy wykonywaniu dostępów do elementów zbioru?

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?

Zadanie 9. Pracujesz nad programem wyświetlającym animacje na ekranie o rozmiarze 640x480 pikseli. Komputer dysponuje 32KB pamięci podręcznej z mapowaniem bezpośrednim, o rozmiarze bloku 8 bajtów. Używasz następujących struktur danych.

```
struct pixel {
  char r;
  char g;
  char b;
  char a;
};
struct pixel buffer[480][640];
int i, j;
char *cptr;
int *iptr;
```

Zakładamy, że `sizeof(char) = 1`, `sizeof(int) = 4`, tablica `buffer` znajduje się w pamięci pod adresem 0x0, pamięć podręczna jest początkowo pusta. Ponadto, dostępy do pamięci dotyczą jedynie elementów tablicy `buffer`. Wszystkie pozostałe zmienne kompilator umieścił w rejestrach. Pamięć podręczna działa w trybie write-back write-allocate.

Jaki jest współczynnik trafień przy wykonaniu poniższego kodu?

```
for (j = 639; j >= 0; j--) {
   for (i = 479; i >= 0; i--) {
       buffer[i][j].r = 0;
       buffer[i][j].g = 0;
       buffer[i][j].b = 0;
       buffer[i][j].a = 0;
   }
}
```

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

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

```
int *iptr = (int *)buffer;
while (iptr < ((int *)buffer + 640*480)) {
     *iptr = 0;
      iptr++;
}
```

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