Imię i nazwisko: _____ Numer indeksu: _____

Architektury systemów komputerowych

19 czerwca 2019

czas trwania: 180 minut

Punkty	0 – 49	50 – 59	60 – 69	70 – 79	80 – 89	90 – 100
Ocena	2.0	3.0	3.5	4.0	4.5	5.0

Zadanie 1 (6). Dla zadanych liczb zmiennopozycyjnych w formacie `1:3:4` (tj. znak, wykładnik, mantysa) oblicz wartość wyrażeń $(a+b)+c$ i $a+(b+c)$. Wynik wyznacz używając zaokrąglania *round-to-even*. Wynik obliczeń należy podać w systemie binarnym. Obliczenia pośrednie wykonaj w brudnopisie.

zmienna	binarnie
a	0 100 1000
b	0 001 1100
c	0 010 0110

$a+b$	=	0b	0	1	0	0	1	1	0	0
$(a+b)+c$	=	0b	0	1	0	1	0	0	0	1
$b+c$	=	0b	0	0	1	1	0	0	1	0
$a+(b+c)$	=	0b	0	1	0	1	0	0	0	0

Zadanie 2 (10). Funkcja `cnb` (ang. *count nonzero bytes*) przyjmuje argument długości słowa maszynowego i wyznacza liczbę jego niezerowych bajtów. Uzupełnij poniższe prostokąty tak, by powstała poprawna definicja funkcji `cnb`.

Przykład: `cnb(0x070050fa00010034) = 5`, `cnb(0x8040201008040201) = 8`.

Najpierw zdefiniuj funkcję pomocniczą `fnb` (ang. *find nonzero bytes*), która każdy niezerowy bajt słowa zamieni na bajt o wartości 1, w przeciwnym wypadku nie zamieni jego wartości.

Przykład: `fnb(0x070050fa00010034) = 0x0100010100010001`

W wyrażeniach poza zmienną `x` i stałymi możesz użyć wyłącznie operatorów bitowych oraz operatora dodawania i przypisania. Każdy z prostokątów powinien zawierać ciąg co najwyżej czterech instrukcji. Łączna liczba wystąpień operatorów nie może przekraczać 21!

```
uint64_t fnb(uint64_t x) {
  x |= x >> 1;
  x |= x >> 2;
  x |= x >> 4;
  x &= 0x0101010101010101;
  return x;
}

unsigned cnb(uint64_t x) {
  x = fnb(x);
  x += x >> 8;
  x += x >> 16;
  x += x >> 32;
  x &= 15;
  return x;
}
```

Wskazówka: W rozwiązaniu wzorcowym `fnb` i `cnb` używają po 7 operatorów.

Zadanie 3 (8). Dla zmiennych x i y typu `long` definiujemy funkcję `min` (ang. *minimum*) następująco:

$$\texttt{min}(x, y) = \begin{cases} x, & \text{gdy } x \leq y; \\ y, & \text{w p.p.} \end{cases}$$

W puste pola wpisz odpowiednie wyrażenia tak, by wartość zwracana przez funkcję `min` była zgodna z powyższą definicją. W wyrażeniach poza stałymi i zdefiniowanymi zmiennymi możesz użyć wyłącznie operatorów bitowych. Łączna liczba wystąpień operatorów nie może przekraczać 15!

Jako krok pośredni obliczeń należy wyznaczyć wartość zmiennej `mask` $= \begin{cases} \texttt{ULONG\_MAX}, & \text{gdy } x \leq y; \\ 0, & \text{w p.p.} \end{cases}$

```
long min(long x, long y) {
  long diff = x - y;
  long mask = ((x & ~y) | ((~x ^ y) & diff)) >> 63 ;
  return (x & mask) | (y & ~mask) ;
}
```

Wskazówka: W rozwiązaniu wzorcowym użyto 11 operatorów.

Zadanie 4 (10). Posługując się `ABI` dla architektury `x86-64` wyznacz rozmiar struktury `node`, rozmiary pól i ich przesunięcie względem początku struktury. W **pierwszą** kolumnę po lewej stronie wpisz **przesunięcie**, a w **drugą rozmiar** danego pola. W kratkę po prawej stronie należy wpisać zoptymalizowaną wersję struktury i jej rozmiar.

```
          struct node {
0   | 4   |   short flags;
8   | 8   |   struct node *next;
16  | 3   |   char key[3];
          |    union {
          |      struct {
24  | 4   |        unsigned n_left;
28  | 4   |        unsigned n_right;
          |      };
          |      struct {
24  | 1   |        unsigned char l_type;
32  | 8   |        char *l_name;
40  | 2   |        short l_value;
          |      };
          |    };
          };

/* sizeof(struct node) == 48 */
```

```
struct node_opt {
    struct node_opt *next;     //  0  8
    union {
      struct {
        unsigned n_left;       //  8  4
        unsigned n_right;      // 12  4
      };
      struct {
        char *l_name;          //  8  8
        unsigned char l_type;  // 16  1
        char key[3];           // 17  3
        short l_value;         // 20  2
        short flags;           // 22  2
      };
    };
};

/* sizeof(struct node_opt) == 24 */
```

Imię i nazwisko: _____ Numer indeksu: _____

Zadanie 5 (13). W prostokąt poniżej wpisz treść procedry w języku C, która wykonuje to samo obliczenie, co poniższa procedura `foo` zaprogramowana w asemblerze. Kod w języku C może zawierać tylko instrukcje sterujące «`for`» i «`if`». Użycie «`goto`» i «`while`» jest niedozwolone. Znak spacji w standardzie ASCII koduje się liczbą 32.

```
foo:
      mov    $32, %cl
      xor    %eax, %eax
.L2:  mov    (%rdi), %dl
      test   %dl, %dl
      je     .L6
      cmp    $32, %cl
      jne    .L3
      xor    %ecx, %ecx
      cmp    $32, %dl
      setne  %cl
      add    %rcx, %rax
.L3:  inc    %rdi
      mov    %dl, %cl
      jmp    .L2
.L6:  ret
```

```
long foo(const char *s) {
  long count = 0;
  char prev = ' ';

  for (char curr; (curr = *s); prev = curr, s++)
    if (prev == ' ')
      count += (curr != ' ');

  return count;
}
```

W prostokąt poniżej wpisz słowne wyjaśnienie, co robi funkcja `foo`.

Funkcja `foo` zlicza w ciągu znaków «`s`» zakończonych zerem liczbę słów niezawierających spacji.

Wskazówka: Rejestry `%cl` i `%dl` to najmłodsze bajty odpowiednio rejestrów `%rcx` i `%rdx`.

Zadanie 6 (8). Przeczytaj poniższy kod w języku `C` i odpowiadający mu kod w asemblerze `x86-64`, po czym wywnioskuj rozmiar struktur `SA` i `SB` oraz wartość stałych `N` i `M`. W kratce poniżej należy umieścić **zwięzły** opis wnioskowania prowadzący do odpowiedzi.

```
typedef struct {
  int s[P];
  int z;
} SA;

typedef struct {
  SA   t[Q];
  long k;
  long y;
} SB;

int foo(SB *p, long i) {
  return p[p->y].t[i].z;
}
```

```
foo:
  mov  0xf8(%rdi),%rax
  shl  $0x8,%rax
  add  %rax,%rdi
  lea  (%rsi,%rsi,2),%rax
  shl  $0x4,%rax
  mov  0x2c(%rdi,%rax,1),%eax
  retq
```

P = 11

Q = 5

sizeof(SA) = 48

sizeof(SB) = 256

Zadanie 7 (12). System posiada 2 KiB sekcyjno-skojarzeniowej dwudrożnej pamięci podręcznej danych. Rozmiar bloku wynosi 8 bajtów. Polityka zastępowania to LRU (ang. *least recently used*). Tablica `A` jest umieszczona w pamięci pod adresem `0x80000` i ma 512 elementów typu «`double`». Przed wykonaniem programu pamięć podręczna danych jest pusta. Jedyną instrukcją generującą dostęp do pamięci danych jest odczyt z tablicy `A`. Niech chybienie zastępujące ma miejsce wówczas, gdy jest związane z usunięciem ofiary ze zbioru, a niezastępujące, gdy blok zostaje skopiowany do nieużywanej linii pamięci podręcznej.

```
double calc(double *A, long si, long di) {
  float r = 0.0;
  for (int i = si; i < 512; i += di)
      r += A[i];
  return r;
}
```

Wykonujemy procedury z poniższej tabelki. Każde kolejne wywołanie procedury widzi stan pamięci podręcznej po wykonaniu poprzedniej procedury. Chcemy znać liczbę trafień i chybień jakie wygeneruje każde z wywołań.

Wywołanie procedury	Trafienia	Chybienia zastępujące	Chybienia niezastępujące
`calc(A, 384, 1);`	0	0	128
`calc(A, 256, 2);`	64	0	64
`calc(A, 193, 2);`	32	64	64

Zadanie 8 (6). Wyznacz zawartość tablicy symboli, tj. zasięg widoczności (`local`, `global`) i sekcję (`.text`, `.data`, `.bss`, `.rodata`, `COMMON`, `UNDEF`) danego symbolu. Podkreśl w kodzie miejsca wystąpienia relokacji.

```
int counters[4];
bool all_ok = true;

static int warns(void) {
  static int *times = &counters[2];
  return *times++;
}

void check(const char *result,
           int (*exit)(int))
{
  const char *str = "warning";
  if (strstr(result, str) == 0)
    if (warns() > 20) {
      all_ok = false;
      exit(1);
    }
}
```

Symbol	Zasięg	Sekcja
warns	local	.text
times	local	.data
‘warning‘	local	.rodata
check	global	.text
all_ok	global	.data
counters	global	COMMON

UWAGA! W kodzie występują stałe, które kompilator umieści w sekcji `.rodata` i przypisze im symbol.

Imię i nazwisko: _____ Numer indeksu: _____

Zadanie 9 (10). TLB jest zorganizowany jako dwudrożna pamięć sekcyjno-skojarzeniowa o 4 zbiorach. Wirtualna przestrzeń adresowa ma 2^{16} bajtów, a fizyczna 2^{14}. Rozmiar strony to 256 bajtów. Polityka wymiany wpisów to NRU (ang. *not recently used*). Pole NRU równe 0 i 1 wyznacza kandydata do usunięcia na odpowiednio lewy i prawy element w zbiorze. Poniżej widnieje pierwotny stan TLB i 16 pierwszych wpisów tablicy stron. Pozostałe wpisy tablicy stron są puste. Procedura obsługi błędu stron ma do dyspozycji kolejne wolne numery stron fizycznych: 08, 14, 06, 1C.

SET	TAG	PPN	TAG	PPN	NRU
0	0A	2C	19	1F	1
1	01	16	-	-	1
2	-	-	-	-	0
3	03	0D	-	-	1

VPN	PPN	VPN	PPN	VPN	PPN	VPN	PPN
00	28	04	31	08	13	0C	19
01	-	05	16	09	17	0D	2D
02	33	06	-	0A	09	0E	11
03	02	07	-	0B	-	0F	0D

Przetłumacz adresy wirtualne podane w poniższej tabeli. Dla każdego adresu wskaż czy chybił w TLB lub wygenerował błąd strony. Podaj też ostateczny stan TLB po przetłumaczeniu wszystkich adresów. **Udzielając odpowiedzi należy użyć zapisu szesnastkowego!** Pierwszy adres został już przetłumaczony, a modyfikacja stanu TLB została uwzględniona w tabelce wyżej. Wszystkie liczby podano w systemie szesnastkowym!

Virt	Phys	Index	Tag	Miss	Fault
2824	2c24	0	0A	NIE	NIE
3226	0826	2	0C	TAK	TAK
64ce	1fce	0	19	NIE	NIE
0e16	1116	2	03	TAK	NIE
088e	138e	0	02	TAK	NIE

SET	TAG	PPN	TAG	PPN	NRU
0	02	13	19	1F	1
1	01	16	-	-	1
2	0C	08	03	11	0
3	03	0D	-	-	1

Uwagi do zadań 10–15: Zdania prawdziwe oznacz literą T, a fałszywe literą N. Funkcja p, która bierze liczbę poprawnie udzielonych odpowiedzi i przydziela punkty, wyraża się następująco: $p(4) = 3$, $p(3) = 2$, $p(2) = 1$, $p(1) = 0$, $p(0) = 0$.

Zadanie 10 (3). Konsolidacja i ładowanie.

- N Jednym z zadań konsolidatora jest tworzenie relokacji w pliku relokowalnym.
- T Konsolidator dynamiczny może działać również po załadowaniu programu do pamięci.
- T Proces nie musi zostać załadowany w całości do pamięci, żeby być wykonywanym.
- N Sekcje programu wykonywalnego przechowują adres pod jaki zostaną załadowane do przestrzeni adresowej.

Zadanie 11 (3). Bezpieczeństwo aplikacji.

- T Randomizacja przestrzeni adresowej znacznie utrudnia przeprowadzanie ataków przez wstrzyknięcie kodu.
- N Dzięki przepełnieniu bufora składowanego na stercie można nadpisać adres powrotu z procedury.
- N Żeby przygotować atak ROP haker powinien przeszukać całą przestrzeń adresową w poszukiwaniu gadżetów.
- T Kanarki to losowe wartości umieszczane na stosie, które sprawdza się w trakcie powrotu z procedury.

Zadanie 12 (3). Tryby pracy procesora, wyjątki i przerwania.

- N Wykonanie instrukcji wywołania systemowego (ang. *syscall*) przełącza procesor w tryb użytkownika.
- T Przy pomocy instrukcji pułaki debugger może ustawić punkt wstrzymania (ang. *breakpoint*).
- N Procesor pobiera adres procedury obsługi wyjątku z tablicy przechowywanej w pamięci programu użytkownika.
- T Próba wykonania instrukcji uprzywilejowanej w trybie użytkownika zakończy się wysłaniem sygnału do procesu.

Zadanie 13 (3). Stronicowanie, translacja adresów i TLB.

- T Procesor może zmieniać rejestr wskaźnika tablicy stron tylko wtedy, gdy pracuje w trybie uprzywilejowanym.
- N Zasięg TLB (ang. *TLB reach*) wyznacza maksymalny rozmiar zbioru roboczego programu.
- T Wymiana stron na dysk jest metodą zmiejszania zbioru rezydentnego programu.
- T W trakcie zmiany przestrzeni adresowej należy wyczyścić TLB, gdyż może przechowywać niewłaściwe wpisy.

Zadanie 14 (3). Hierarchia pamięci.

- N W wieloplaterowych dyskach magnetycznych wszystkie głowice poruszają się niezależnie.
- T Czas odczytu losowych lokacji pamięci DRAM jest zdominowany przez czas zmiany wiersza.
- N DMA to mechanizm sprzętowy kopiowania pamięci urządzenia do pamięci podręcznej procesora.
- T Zapis strony na dysku półprzewodnikowym SSD musi być poprzedzony wymazaniem całego bloku.

Zadanie 15 (3). Mikroarchitektura procesora.

- T Procesor może w jednym cyklu zegarowym wykonać wiele niezależnych od siebie instrukcji.
- T Dzięki predyktorowi skoków procesor może przeprowadzać spekulatywne wykonywanie kodu.
- T Procesory nie muszą wykonywać instrukcji programu sekwencyjnie.
- T Tempo wykonania instrukcji jest głównie ograniczone liczbą instrukcji odczytów z pamięci.

Imię i nazwisko: _____ Numer indeksu: _____