Imię i nazwisko: _____ Numer indeksu: _____

Architektury systemów komputerowych

10 września 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 (7). 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	0 101 0001
b	1 100 1101
c	1 001 0100

$a+b$ = 0b | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 |

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

$b+c$ = 0b | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 0 |

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

Zadanie 2 (10). W kodowaniu liczb BCD (ang. *binary-coded decimal*) każde kolejne 4 bity liczby kodują kolejne cyfry dziesiętne. Np. liczba 123 jest kodowana jako `0x123`. Wiele operacji na liczbach BCD można wykonać za pomocą prostych operacji bitowych. Aby podwoić taką liczbę wystarczy dodać 3 do wszystkich cyfr większych od 4, a następnie całość przesunąć bitowo o jeden. Uzupełnij poniższe prostokąty, tak by powstała poprawna definicja funkcji `bcdx2`, która podwaja liczbę w kodowaniu BCD.

Przykład: `bcdx2(0x0700508900010034) = 0x1401017800020068`.

Najpierw zdefiniuj funkcję pomocniczą `bcdgt4`, która cyfry większe od 4 zamieni na 1, a pozostałe cyfry zamieni na 0. Nie przejmuj się cyframi większymi niż 9, bo nie są to poprawne cyfry dziesiętne.

Przykład: `bcdgt4(0x0732580123946645) = 0x0100110000101101`.

W wyrażeniach poza zmiennymi `x` i `y` oraz 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 18!

```
uint64_t bcdgt4(uint64_t x) {
  x += 0x3333333333333333;
  x >>= 3;
  x &= 0x1111111111111111;
  return x;
}

uint64_t bcdx2(uint64_t x) {
  uint64_t y = bcdgt4(x);
  x += y | (y << 1);
  x <<= 1;
  return x;
}
```

Wskazówka: W rozwiązaniu wzorcowym `bcdgt4` i `bcdx2` używają odpowiednio po 3 operatory i 4 operatory.

Zadanie 3 (8). Dla zmiennych x i y typu `long` funkcja `tryadd` oblicza sumę x i y, ale gdy to dodawanie generuje przepełnienie, to zwraca wartość x. Formalnie, funkcja `tryadd` jest zdefiniowana następująco:

$$\texttt{tryadd}(x,y) = \begin{cases} x+y, & \text{gdy } \texttt{LONG\_MIN} \le x+y \le \texttt{LONG\_MAX}; \\ x, & \text{w p.p.} \end{cases}$$

W puste pola wpisz odpowiednie wyrażenia tak, by wartość zwracana przez funkcję `tryadd` była zgodna z powyższą definicją. W wyrażeniach poza stałymi i zdefiniowanymi zmiennymi możesz użyć wyłącznie dodawania i operatorów bitowych. Łączna liczba wystąpień operatorów nie może przekraczać 15!
Jako krok pośredni obliczeń należy wyznaczyć wartość zmiennej `mask` zdefiniowanej następująco:

$$\texttt{mask} = \begin{cases} \texttt{ULONG\_MAX}, & \text{gdy } \texttt{LONG\_MIN} \le x+y \le \texttt{LONG\_MAX}; \\ 0, & \text{w p.p.} \end{cases}$$

```
long tryadd(long x, long y) {
  long sum = x + y;
  long mask = [((x ^ y) | (x ^ ~sum)) >> 63];
  return [x + (mask & y)];
}
```

Wskazówka: W rozwiązaniu wzorcowym użyto 7 operatorów.

Zadanie 4 (10). Przeczytaj poniższy kod w języku `C` i odpowiadający mu kod w asemblerze `x86-64`, po czym wywnioskuj wartość stałych `L`, `M` i `N`. W kratce poniżej należy umieścić **krótki i zwięzły** opis wnioskowania prowadzący do odpowiedzi. Tablice są umieszczone w pamięci w porządku row-major.

```
long array1[L][M][N];
long array2[M][N][L];

void copy(long i, long j, long k)
{
  array1[i][j][k] =
    array2[j][k][i];
}
```

```
copy:
  lea   (%rsi,%rsi,4), %rax
  lea   (%rsi,%rax,2), %rcx
  lea   (%rdi,%rdi,4), %rax
  lea   (%rdi,%rax,2), %rax
  lea   (%rax,%rax,4), %rax
  add   %rcx, %rax
  add   %rdx, %rax
  add   %rdx, %rcx
  lea   (%rdi,%rcx,2), %rdx
  mov   array2(,%rdx,8), %rdx
  mov   %rdx, array1(,%rax,8)
  ret
```

L = 2

M = 5

N = 11

Imię i nazwisko: _____ Numer indeksu: _____

Zadanie 5 (8). 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   | 2     char id[2];
8   | 8     int (*hashfn)(char *);
16  | 2     short flags;
           union {
             struct {
20  | 2        short n_key;
24  | 8        int n_data[2];
32  | 1        unsigned char n_type;
             };
20  | 8          unsigned l_value[2];
           };
         };
```

```
struct node_opt {
    int (*hashfn)(char *);     //  0  8
    short flags;               //  8  2
    char id[2];                // 10  2
    union {
      struct {
        int n_data[2];         // 12  8
        short n_key;           // 20  2
        unsigned char n_type;  // 22  1
      };
      unsigned l_value[2];     // 12  8
    };
};
```

`/* sizeof(struct node) ==` 40 `*/`

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

Zadanie 6 (10). 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. Parametr «s» wskazuje na ciąg 7-bitowych kodów ASCII.

```
foo:  xor   %eax, %eax
      xor   %ecx, %ecx
.L2:  mov   (%rdi), %dl
      test  %dl, %dl
      je    .L7
      inc   %rax
      cmp   %dl, %cl
      jle   .L3
      mov   $1, %eax
.L3:  inc   %rdi
      mov   %edx, %ecx
      jmp   .L2
.L7:  ret
```

```
long foo(const char *s) {
  char curr, prev = 0;
  long run = 0;

  for (; (curr = *s); prev = curr, s++) {
    if (prev <= curr)
      run++;
    else
      run = 1;
  }
  return run;
}
```

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

«foo» wyznacza w ciągu znaków «s» zakończonym zerem maksymalną długość sufiksu składającego z niemalejących kodów ASCII.

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

Zadanie 7 (9). Niech `x` i `y` będą liczbami całkowitymi typu `int`. Podaj dowolne wartości, dla których poniższe wyrażenia będą fałszywe lub napisz prawda, jeśli wyrażenie jest zawsze prawdziwe.

`(x ^ y) < 0`	x=y=0
`((~(x | (~x + 1)) >> 31) & 0x1) == !x`	PRAWDA
`(x ^ (x>>31)) - (x>>31) > 0`	x=0
`((x >> 31) + 1) >= 0`	PRAWDA
`(!x | !!y) == 1`	x=1, y=0
`x ^ y ^ (~x) - y == y ^ x ^ (~y) - x`	x=1, y=0

Zadanie 8 (8). Na podstawie poniższego kodu w asemblerze x86-64 uzupełnij (a) w kodzie źródłowym w języku C puste pola, w których brakuje słowa kluczowego `break` lub specjalnej dyrektywy `fallthrough` (dostępne w `gcc-8`) oznaczającej przejście do wykonania następnego przypadku (b) tabelę skoków, która widnieje po prawej stronie, używaną przez instrukcję pod adresem 0x401b5a.

```
long lol(long a, long b)
{
 switch (a)
 {
  case 210:
   b *= 13;
   break;
  case 213:
   b = 18243;
   fallthrough;
  case 214:
   b *= b;
   break;
  case 216:
  case 218:
   b -= a;
   break;
  case 219:
   b += 13;
   fallthrough;
  default:
   b -= 9;
  }
 return b;
}
```

```
401b4d <lol>:
401b4d: lea  -210(%rdi),%rax
401b54: cmp  $9,%rax
401b58: ja   401b82 <lol+0x35>
401b5a: jmp *0x47d008(,%rax,8)
401b61: lea  (%rsi,%rsi,2),%rax
401b65: lea  (%rsi,%rax,4),%rax
401b69: ret
401b6a: mov  $18243,%esi
401b6f: mov  %rsi,%rax
401b72: imul %rsi,%rax
401b76: ret
401b77: mov  %rsi,%rax
401b7a: sub  %rdi,%rax
401b7d: ret
401b7e: add  $0xd,%rsi
401b82: lea  -9(%rsi),%rax
401b86: ret
```

0x47d008:	0x401b61
0x47d010:	0x401b82
0x47d018:	0x401b82
0x47d020:	0x401b6a
0x47d028:	0x401b6f
0x47d030:	0x401b82
0x47d038:	0x401b77
0x47d040:	0x401b82
0x47d048:	0x401b77
0x47d050:	0x401b7e

Imię i nazwisko: _____ Numer indeksu: _____

Zadanie 9 (12). System posiada 1 KiB sekcyjno-skojarzeniowej czterodrożnej pamięci podręcznej danych. Rozmiar bloku wynosi 16 bajtów. Polityka zastępowania to LRU (ang. *least recently used*). Dwuwymiarowa tablica `A` jest umieszczona w pamięci pod adresem `0x80000` i zawiera 64 wiersze po 64 elementy typu «`int`». 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.

```
int maxH(int A[N][N], int H, int W) {
  int m = 0;
  for (int i = 0; i < H; i++)
    for (int j = 0; j < W; j++)
      m = max(m, A[i][j]);
  return m;
}
```

```
int maxV(int A[N][N], int H, int W) {
  int m = 0;
  for (int j = 0; j < W; j++)
    for (int i = 0; i < H; i++)
      m = max(m, A[i][j]);
  return m;
}
```

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
`maxH(A, 2, 64);`	96	0	32
`maxV(A, 64, 2);`	2	124	2
`maxH(A, 64, 64);`	3072+30	4+960	30
`maxV(A, 64, 64);`	0	4096	0

Zadanie 10 (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.

```
static int array[100];
int *iptr = &array[20];
int bias;

static int map(int (*fn)(int),
               int a[]) {
  for (int sum = 0;
       *a;
       sum += fn(*a++));
  return sum + bias;
}

void foo() {
  printf("n = %d\n",
         map(dbl, iptr));
}
```

Symbol	Zasięg	Sekcja
array	local	.bss
iptr	global	.data
bias	local	COMMON
map	local	.text
foo	global	.text
printf	global	UNDEF
'...'	local	.rodata
dbl	global	UNDEF

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

Zadanie 11 (12). TLB zorganizowano jako dwudrożną pamięć sekcyjno-skojarzeniową o 4 zbiorach. Wirtualna przestrzeń adresowa ma 2^{14} bajtów, a fizyczna 2^{12}. Rozmiar strony to 64 bajty. 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 początkowy stan TLB i 16 pierwszych wpisów tablicy stron (pozostałe 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	03	19	-	-	1
1	1A	1C	02	17	1
2	3F	3F	-	-	1
3	-	-	-	-	0

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	VPN	PPN	Index	Tag	Miss	Fault
1A7C	73C	69	1C	1	1A	NIE	NIE
03E1	361	0F	0D	3	03	TAK	NIE
1964	224	65	08	1	19	TAK	TAK
0240	5C0	09	17	1	02	TAK	NIE
3FAA	FEA	FE	3F	2	3F	NIE	NIE

SET	TAG	PPN	TAG	PPN	NRU
0	03	19	-	-	1
1	02	17	19	08	1
2	3F	3F	-	-	1
3	03	0D	-	-	1