Imię i nazwisko: _____ Numer indeksu: _____

# Architektury systemów komputerowych

*21 czerwca 2016*

*czas trwania: 150–180 minut*

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

**Uwagi do zadań 1–2:** Należy używać wyłącznie operatorów arytmetyczno-logicznych, przypisania, stałych i dodatkowych zmiennych. **Zabrania się** korzystania z instrukcji sterowania, w tym instrukcji i operatora warunkowego, oraz operatorów mnożenia, dzielenia i modulo. **Nie można** dopuścić do wystąpienia nadmiaru i niedomiaru!

**Zadanie 1 (6).** Przetłumacz poniższą funkcję na procedurę języka `C` o sygnaturze `int num(unsigned)`.

$$\texttt{num}\,(n) = \begin{cases} i & \text{dla } n = 2 \cdot i \\ -i & \text{dla } n = 2 \cdot i + 1 \end{cases}$$

**Zadanie 2 (6).** Przetłumacz poniższą funkcję na procedurę języka `C` o sygnaturze `int avg(int, int)`.

$$\texttt{avg}\,(x, y) = \left\lfloor \frac{x + y}{2} \right\rfloor$$

**Zadanie 3 (8).** Przetłumacz kod procedury `foobar` z asemblera `x86-64` do języka `C`. **Należy posłużyć się** wysokopoziomowymi instrukcjami sterującymi. Używanie etykiet i instrukcji `goto` jest **niedozwolone**.

```
1  foobar:
2        mov   %rdi, %rax
3        xor   %rdx, %rdx
4  .L2:  cmp   $0, (%rax)
5        je    .L7
6        jg    .L3
7        dec   %rdx
8        jmp   .L4
9  .L3:  inc   %rdx
10 .L4:  add   $8, %rax
11       jmp   .L2
12 .L7:  mov   %rdx, (%rsi)
13       ret
```

**Zadanie 4 (10).** Przeczytaj poniższy kod w języku `C` i odpowiadający mu kod w asemblerze `x86-64`, po czym wywnioskuj rozmiar struktur `strA` i `strB` oraz wartość stałych `N` i `M`. W kratce poniżej należy umieścić **zwięzły** opis wnioskowania prowadzący do odpowiedzi.

```
1  typedef struct {
2    int  s[M];
3    long z;
4  } strA;
5
6  typedef struct {
7    strA  t[N];
8    long  k;
9    short x;
10   short y;
11 } strB;
12
13 long foo(strB *bp, long i) {
14   return bp[bp->k].t[i].z;
15 }
```

```
1  foo:
2      movq  120(%rdi), %rax
3      movq  %rax, %rdx
4      salq  $7, %rdx
5      leaq  (%rdx,%rax,8), %rax
6      addq  %rax, %rdi
7      leaq  (%rsi,%rsi,2), %rax
8      movq  16(%rdi,%rax,8), %rax
9      ret
```

Imię i nazwisko: _____ Numer indeksu: _____

**Zadanie 5 (6).** Niech `i` będzie niezerową liczbą całkowitą typu `int`, oraz `f` i `g` niezerowymi liczbami zmiennopozycyjnymi typu `float`. Podaj dowolne wartości, dla których poniższe wyrażenia będą prawdziwe:

```
    i * i < i          _____
  i >> 2 != i / 4      _____
f / 2.0 == f / -2.0    _____
(f + g) / (f + g) != 1.0 _____
```

**Zadanie 6 (10).** TLB jest zorganizowany jako czterodrożna pamięć sekcyjno-skojarzeniowa o 4 zbiorach. Wirtualna przestrzeń adresowa ma $2^{14}$ bajtów, a fizyczna $2^{12}$. Rozmiar strony to 64 słowa. Polityka wymiany wpisów to *least recently used* – im wyższy numer znacznika LRU tym wpis jest starszy. Poniżej podano pierwotny stan TLB i 16 pierwszych wpisów tablicy stron. **Dane są w zapisie szesnastkowym!**

| SET | TAG | PPN | LRU | TAG | PPN | LRU | TAG | PPN | LRU | TAG | PPN | LRU |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 07 | 02 | 2 | 09 | 0D | 1 | 00 | 28 | 0 | - | - | 3 |
| 1 | 03 | 2D | 0 | - | - | 3 | - | - | 3 | - | - | 3 |
| 2 | - | - | 3 | - | - | 3 | - | - | 3 | - | - | 3 |
| 3 | 1A | 34 | 0 | - | - | 3 | - | - | - | - | - | 3 |

| VPN | PPN | VPN | PPN |
|---|---|---|---|
| 0 | 28 | 8 | 13 |
| 1 | - | 9 | 17 |
| 2 | 33 | A | 09 |
| 3 | 02 | B | - |
| 4 | 31 | C | 19 |
| 5 | 16 | D | 2D |
| 6 | - | E | 11 |
| 7 | - | F | 0D |

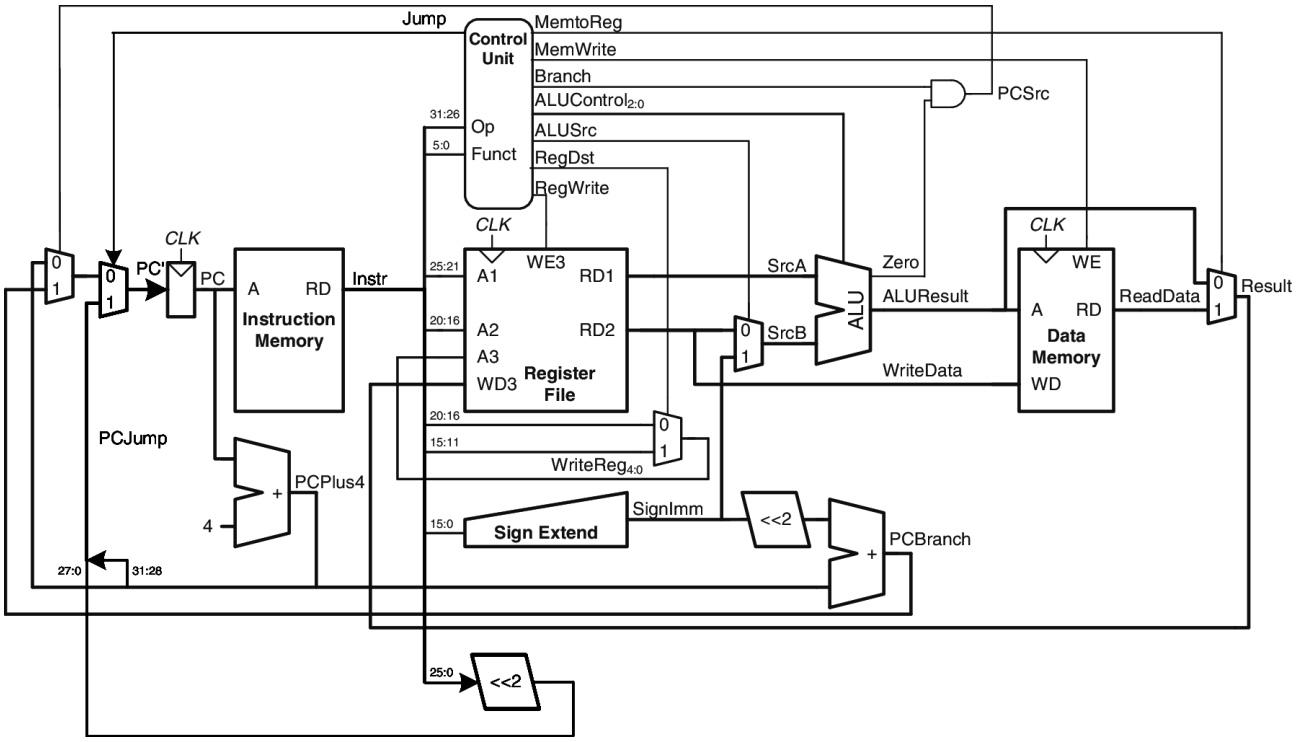
Przetłumacz adresy wirtualne podane w poniższej tabeli. Dla każdego adresu wskaż czy chybił w TLB lub wygenerował błąd strony. Podaj też ostateczny stan TLB po przetłumaczeniu wszystkich adresów. **Udzielając odpowiedzi należy użyć zapisu szesnastkowego!** Pierwszy adres został już przetłumaczony, a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

| adres wirtualny | adres fizyczny | indeks TLB | znacznik TLB | chybienie w TLB | błąd strony |
|---|---|---|---|---|---|
| 0024 | a24 | 0 | 00 | TAK | NIE |
| 0326 | | | | | |
| 1ace | | | | | |
| 0216 | | | | | |
| 018e | | | | | |
| 032a | | | | | |

| SET | TAG | PPN | LRU | TAG | PPN | LRU | TAG | PPN | LRU | TAG | PPN | LRU |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | | | | | | | | | | | | |
| 1 | | | | | | | | | | | | |
| 2 | | | | | | | | | | | | |
| 3 | | | | | | | | | | | | |

**Zadanie 7 (8).** Na poniższym schemacie widnieje jednocyklowa implementacja procesora MIPS. Zaproponuj kodowanie instrukcji, nowe sygnały kontrolne i modyfikacje schematu niezbędne do obsługi dodatkowej instrukcji. Podaj stan wszystkich sygnałów sterujących. Modyfikowanie ALU jest **zabronione**!

**UWAGA:** Na schemacie należy jedynie zakreślić modyfikowany fragment procesora i przerysować go ze zmianami do kratki!




| mnemonik | typ | semantyka |
|---|---|---|
| lwx $Rd,$Rs[$Rt] | R | Reg[Rd] := Mem[Reg[Rs] + Reg[Rt] * 4] |

| | MemToReg | MemWrite | Branch | $ALUControl_{2:0}$ | ALUSrc | RegDst | RegWrite | Jump |
|---|---|---|---|---|---|---|---|---|
| lwx | | | | | | | | |

Imię i nazwisko: _____ Numer indeksu: _____

**Zadanie 8 (8).** Rozważmy potokowy procesor MIPS z implementacją obejść i obsługą hazardów, ale bez *branch delay slots*. Zakładamy, że skoki są obliczane w fazie ID, a wykonują się w etapie EX. Przyjmujemy strategię przewidywania typu „*zawsze nie wykonuj skoku*". Narysuj diagram stanu potoku wykonania poniższego kodu. Wskaż hazardy *Read-After-Write* i oznacz ścieżki przekazywania danych przy pomocy obejść. Zakładamy, że skok w linii 3 nie zostaje wykonany, a w linii 6 zostaje.

```
1       lw     $3,0($5)       IF   ID   EX   MEM   WB
2       srl    $8,$8,1
3       beq    $2,$3,.L1
4       lw     $3,0($6)
5       addi   $3,$3,-16
6 .L1:  bne    $2,$3,.L2
7       subu   $2,$2,$8
8 .L2:  sw     $2,-4($7)
```

**Zadanie 9 (10).** Rozważmy superskalarny procesor MIPS z dwoma potokami: U-pipe, który wykonuje wszystkie instrukcje, oraz V-pipe, który wykonuje wyłącznie operacje arytmetyczno-logiczne (w tym na liczbach zmiennopozycyjnych). Czas przetwarzania instrukcji to: ALU – 1, MEM – 2, FP-ADD – 2 i FP-MUL – 3 cykli, a interwał inicjacji wynosi 1. Ile cykli wymaga przetworzenie jednej iteracji poniższej pętli? Rozwiń ją jednokrotnie po czym zoptymalizuj jej ciało, a następnie podaj liczbę cykli wymaganych do jej wykonania.

```
1 loop:
2     ld.s   $f1,0($a1)   # X[i]
3     mul.s  $f2,$f1,$f0  # a * X[i]
4     ld.s   $f3,0($a2)   # Y[i]
5     add.s  $f3,$f2,$f3  # a * X[i] + Y[i]
6     st.s   $f3,0($a2)   # Y[i] = a * X[i] + Y[i]
7     addi   $a1,$a1,4
8     addi   $a2,$a2,4
9     bne    $a2,$a3,loop # $a3 = koniec tablicy Y
```

| iteracje | wersja pętli | cykli / element |
|---|---|---|
| 1× | oryginalna | |
| 2× | zoptymalizowana | |

**Uwagi do zadań testowych:** Zdania prawdziwe oznacz literą `T`, a fałszywe literą `N`. Liczba przydzielonych punktów jest określona wzorem $\lfloor 2^{p-2} \rfloor$, gdzie $p$ to liczba poprawnie udzielonych odpowiedzi.

**Zadanie 10.** *Application Binary Interface*, konwencja wołania procedur.

- ☐ `ABI` określa metodę wywoływania funkcji jądra systemu operacyjnego.
- ☐ Rejestry służące do przekazywania argumentów mogą być nadpisywane przez funkcję wołaną.
- ☐ Jeśli wynik funkcji nie mieści się w rejestrach, to funkcja wołana musi przygotować miejsce na wynik w swojej ramce stosu.
- ☐ Przed wywołaniem funkcji, adres wierzchołka stosu musi być wyrównany do wielokrotności rozmiaru największego słowa maszynowego.

**Zadanie 11.** Reprezentacja plików wykonywalnych, modułów i bibliotek. Konsolidacja i ładowanie.

- ☐ Punkt wejścia (ang. *entry point*) do programu napisanego w języku `C` wskazuje na adres funkcji `main`.
- ☐ Sekcja `.bss` może mieć przypisane relokacje.
- ☐ W trakcie konsolidacji pliku wykonywalnego sekcje są łączone w segmenty.
- ☐ Plik wykonywalny skonsolidowany dynamicznie nie posiada tablicy symboli.

**Zadanie 12.** Tryby pracy procesora i przerwania.

- ☐ W trakcie obsługi przerwania obsługa wyjątków jest zablokowana.
- ☐ Procesor może przejść do trybu uprzywilejowanego nie używając instrukcji uprzywilejowanych.
- ☐ Procesor odkłada na stos rejestry ogólnego przeznaczenia przed wywołaniem procedury obsługi przerwania.
- ☐ By powrócić do wykonania przerwanego programu, procedura obsługi przerwania musi wykonać specjalną instrukcję uprzywilejowaną.

**Zadanie 13.** Urządzenia i obsługa wejścia–wyjścia.

- ☐ Przerwanie może sygnalizować zdarzenia nadchodzące z więcej niż jednego urządzenia.
- ☐ Komunikacja z użyciem przerwań jest przeważnie wydajniejsza niż odpytywanie.
- ☐ W systemie z `DMA`, po zleceniu odczytu bloku pamięć operacyjna regularnie odpytuje urządzenie i kopiuje z niego dane bez udziału procesora.
- ☐ Wraz z odległością od środka talerza (platera) ilość sektorów na ścieżkę maleje.

**Zadanie 14.** Lokalność i interakcja z pamięcią podręczną.

- ☐ Przeglądanie listy dwukierunkowej charakteryzuje się dobrą lokalnością przestrzenną.
- ☐ Proces generuje dużo błędów stron, jeśli jego zbiór rezydentny jest większy od zbioru roboczego.
- ☐ Przetwarzanie kodu pętli wykazuje dobrą lokalność czasową.
- ☐ Pamięć podręczna z polityką zapisu *write-through* może dawać niepożądane efekty przy zapisie do pamięci, która będzie kopiowana z użyciem `DMA`.

**Zadanie 15.** Stronicowanie, `TLB` i jednostka zarządzania pamięcią (`MMU`).

- ☐ `MMU` procesorów `x86-64` może modyfikować zawartość tablicy stron bez udziału systemu operacyjnego.
- ☐ Dla każdej strony można zadać czy jej zawartość ma być przechowywalna w pamięci podręcznej.
- ☐ Stronicowanie umożliwia zarówno izolację jak i współdzielenie pamięci między procesami.
- ☐ Pamięć `TLB` może być indeksowana i tagowana wyłącznie adresami wirtualnymi.

**Zadanie 16.** Mikroarchitektura procesora.

- ☐ Mechanizm przewidywania skoków swym działaniem obejmuje także instrukcje powrotu z procedury.
- ☐ W procesorze potokowym odrębna pamięć podręczna instrukcji i danych całkowicie eliminuje hazard strukturalny w dostępach do pamięci.
- ☐ Przemianowywanie rejestrów pozwala procesorowi na usuwanie hazardów danych *Read-After-Write*.
- ☐ Procesory *Out-of-Order* pozwalają na zatwierdzanie efektów działania instrukcji w innym porządku niż występują one w programie.