Parking

| | | | | | | | | | | | | | |

Za brak zaparkowanege kursora grozi blokada i wizyta w wujka google w celu uiszczenia kary

OIAK Kolokwium termin 0 => wyniki https://www.facebook.com/groups/175246409273826/509672839164513/

Alternatywne opracowanie zadań 1,2,4:

;Strona 1 (zad 1, 2): https://www.dropbox.com/s/z6kmcfkrfdok681/Strona%201.jpg

Strona 2 (zad 4): https://www.dropbox.com/s/1f30vbux38bkw6m/Strona%202.jpg

//są gdzieś jego wykłady albo jakieś inne prezentacje? czy coś…

//On nie ma swoich wykladow, caly semestr jechał na wykładach od Biernata, **przy czym slajdy po raz pierwszy oglądał na wykładzie…**

//a na tych slajdach jest to co w tym kole?

//Niestety nie. Slajdy biernata mają mało wspólnego nawet z kołami Biernata.

// Podczas kursu nawet nie wspominał o poniższych zagadnieniach.

//No to niefajnie.

// wg mnie z tematem są związane zadania 4 i 5 :D dobre koło

nie mam zielonego pojęcia jak cokolwiek stąd ruszyć, :/ a myślałem że umiem

// apropos oceny końcowej: u ludzi na grupie z 2011 (rok wyzej) krąży plota, że u Biernata przelicznik jest następujący:

if ((ocena_z_lab*100/5.5 + ocena_z_proj*100/5.5 + ile%_z_egz)/3 > *50*)

“zaliczasz”;

else “no trudno”;

W zeszłym roku tak było. Czy w tym też? Opinie są rozbieżne.

załóżmy, że od 50, w razie co będzie miłe rozczarowanie

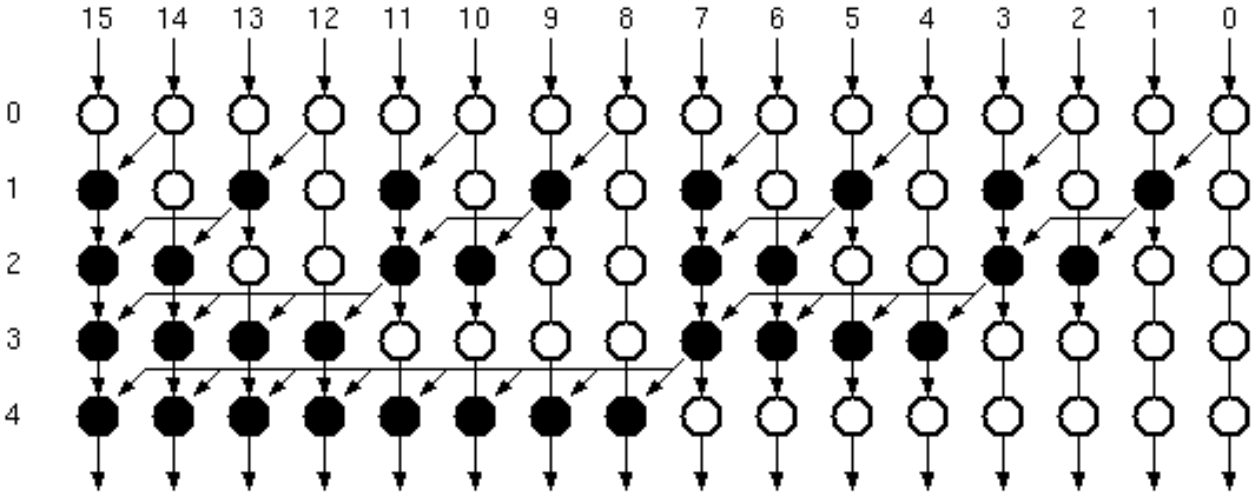
to je patronik nie b’ernat

Treści zadań:

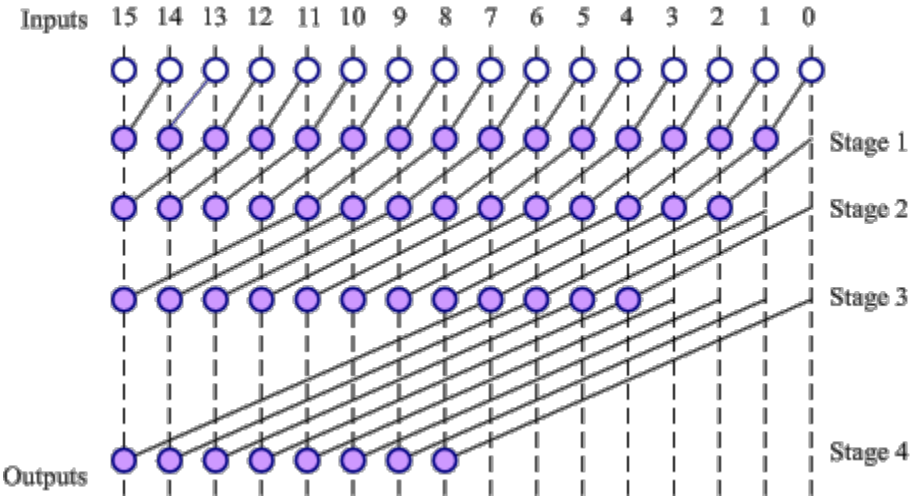
1. Narysuj schemat sumatora prefiksowego d_0%3+3 bitowego w architekturze b_0 = 0 - Kogge'a Stone'a; b_0 =1 Sklanskyego i pokaż ich działanie dla dwóch dowolnych wektorów wejściowych

Sklansky - oczywiscie uciac do zadanej liczby bitow
//a dokładnie co oznacza ten zapis d_0%3+3 ? i co oznacza b_0 = 0 ?
// d0 - najmniej znaczaca cyfra indeksu
// %3 - modulo 3 (reszta z dzielenia przez 3)
// +3 - plus (dodac) 3
// b0 - najmniej znaczacy bit numeru indeksu - b0=0 indeks parzysty, b0=1 nieparzysty
//czyli gdy indeks kończy się na np. 0 to mamy narysować sumator 3-bitowy w arch. Kogge'a Stone'a. czyli z tego rysunku poniżej mają zostać tylko kolumny 0, 1 i 2 ?
// chyba tak
//na pewno tak jest, że się po prostu obcina kolumny

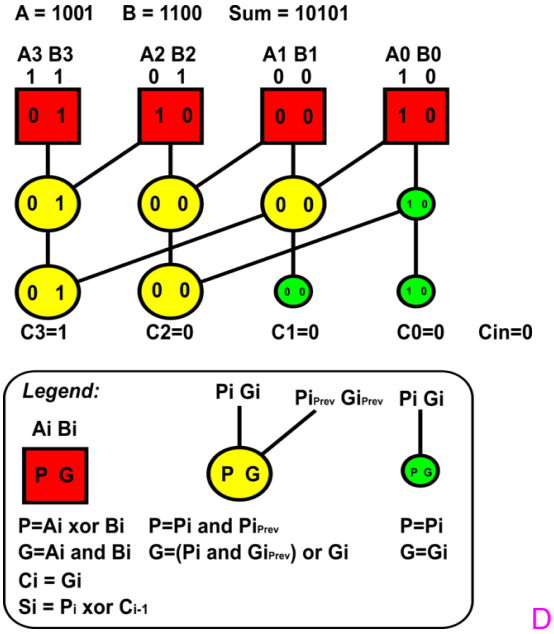
Slansky:



Kogge-Stone:



a jak pokazac przykładowe działanie tego na dowolnych wektorach?
A no np tak:(wzory sie zgadzaja, przerobiłem pare przykładów, dla różnych dł. słów wejściowych)



Dla Sklanskiego wzory zostają takie +same, tylko inny schemat.
Ostatnia suma jaka nalezy obliczyć(dla powyższego przykładu) to S3, a S4 to po prostu przeniesienie z poprzedniej pozycji.
czy w ostatnim elemencie musimy dodać kreskę w przód ? tzn ostatnie przeniesienie ? w przykładzie wyżej na najstarszej pozycji sumujesz 1 + 1 i tracisz wynik bo na tej pozycji zostaje 0 a 1 ma przejsc na następną. Jak nie ma kolejnego wyjścia na starszą pozycję to tracisz najsatrszą pozycje.
Ogólnie tak: oprócz powyższego znalazłem jeszcze inny schemat z obliczonym przykładem i również nie miał on żadnej kreski przy ostatnim elemencie. Teoretycznie mozna dodac taką kreskę, w praktyce nie uważam tego za konieczne, dość intuicyjne jest że na ostatnią pozycję wyskoczy nam przeniesienie z poprzedniej.

//gdyby ktoś się zastanawiał, to przy wzorze na sumę Pi jest P pochodzącym z czerwonego kwadratu a nie z góry (na rysunku z dołu bo rośnie w dół) drzewa jak G w przypadku Ci

2. Dwie liczby zmiennoprzecinkowe, z tą różnicą, że mantysa ma 7 bitów, która jest szesnastkowo zapisana jako A = $h_3h_2h_1h_0$, B = $h_1h_2h_3h_4$. Podaj sumę oraz iloraz tych liczb, podaj wyniki dziesiętnie i szesnastkowo.
//czyli 1 bit to znak, 8 bitów to wykładnik i 7 bitów to mantysa.
Dla np. 198691

		Z	E	M
A:	8691	1	00001101	0010001
B:	9689	1	00101101	0001001

Skorzystajmy z metody dzielenia nieodtwarzającego

```
  (0)1101111|      (0)1110111
  _______(0)|      _________
  (1)0010001|   :  (1)0001001
 +(0)1110111
  (0)0001000
```

poddaje sie to sie robi dotad az sa liczby za skala a tu nie ma zdanych xd

Z- znak
E- wykladnik
M- Mantysa / Po tomczakowemu mnożnik (wg. tomczaka okreslenia mantysa sie juz nie używa)

Tutaj chyba będzie po prostu B, A jest mniejsze od B 2^31-krotnie, więc A nie ma żadnego wpływu. Mam rację?
Schematy ogólne:

-Dzielenie - dzielimy mantysy odejmujemy wykladniki i najprawdopodobniej dodajemy do wykladnika tyle ile trzeba bylo przesunac przecinek aby wyrownac mantyse wynikowa zeby byla postaci 1,1010101

-Mnozenie mnozymy mantysy dodajemy wykladniki i jak wyżej dalszy ciąg
W obydwu przypadkach pamietamy o zamianie przy operacji dodawania i odejmowania na +N czyli negujemy wszystkie bity oprocz najstarszego wykonujemy operacje wynik zamieniamy spowrotem na podstawe 2 czyli negujemy wszystkie bity oprocz najstarszego poraz kolejny. Przy dodawaniu wyrównania juz nie zamieniamy na +N tylko dodajemy w dwojkowym

-Odwrotność - zamieniamy tak jak wyzej z +N na 2 czy na odwrot znaczy tak jak wyzej opisane jest . Po zamianie obliczamy odwrotnosc liczby to chyba kazdy wie jak w u2 zrobic
Znowu zamieniamy na +N
Dzielimy 1 przez mantyse przesuwamy przecinek aby liczba bitów licznika i mianownika byla taka sama. Przesuwamy przecinek w prawo do pierwszej jedynki. Na koniec odejmujemy od wykładnika tyle ile trzeba bylo przesunąć.

Rozwiązanie dzielenia:
teraz trzeba popisac na kartce :D:D

A potrafi ktoś zrobić dodawanie?

W dodawaniu to sie normalizuje do liczby wiekszej, bo ta mniejsza jest mniej istotna.
Wzór jest taki że M1*2^E1 + M2^E2 = 2^E2(M1*2^(E1-E2) + M1) przy zalozeniu ze E1 jest wieksze niz E2. Zaczerpniete z ćwiczeń u Postawki.
Wzór to wiem, też mam nawet od niej notatki. Mam przykład z ćwiczeń, gdzie E2 jest mniejsze od E1 dokładnie o 3 i wtedy po prostu mnożymy mantysę drugiej liczby (wraz z ukrytą jedynką) razy 2^-3, ale w przypadku mojego nru indeksu z dzisiejszego koła dostaje liczbę o 32 mniejszą, to co, 2^-23? Chyba nie za bardzo o tyle przesuwać w lewo.

Nawiązujac do powyższego problemu - przesuniecie o 23 w lewo/prawo, dodanie i powrót do postaci bez przesuniecia nie ma prawa zmienić wyniku który jest zapisany na 8 bitach(zmiana bedzie obserwowana dopiero przy 23 bicie, a po powrocie i zachowaniu precyzji będzie całkowicie niewidoczna). Wobec tego zamiast robić przesunięcie i pisać 30 bitowe liczby można słownie opisać cały proces i wynik takiego dodawania napisać w kolejnej linijce.

OK, to przykładowo może ktoś rozpisze dodanie takich dwóch liczb?
0 00001110 0010101
0 00101110 0000000
S wykładnik mantysa
Jak te wykładniki sie wyrównuje, mają być całkowicie sobie równe (tzn odpowiednie bity mają się zgadzać?). - chyba tak, w sumie napewno tak || no dobra, akurat w podanym przykładzie wykładniki różnię się tylko jednym bitem, na pozycji 5., czyli różnią się o 32. Jak to wyskalować?
Masz pierwsza liczbe: 1,0010101 * 2^(-113) i drugą 1,0000000 * 2^(-81)
Robisz tak: 1,0010101 * 2^(-32) * 2^(-81) z pierwsza i masz 0,0000000 * 2^(-81) (jakoś tak), dalej je sumujesz i wychodzi ci 1,0000000 * 2^(-81) i koniec - **tak mi się wydaje, poprawcie mnie jak źle**
// jak mnozysz przez 2^(-32) to przesuwasz wszystkie bity o 32 pozycje w prawo
Wykładnik jest zapisany w postaci +N, chyba, sam nie wiem już
Zgadza się, ale jakbyśmy chcieli zapisać wynik dwójkowo, to do wykładnika -81 dodajemy obciążenie? - 81 + 127 = 46, i to jest wykładnik wynikowy? Czy liczba wynikowa będzie ujemna (ze znakiem 1) i wykładnikiem o wartości 81?
Nom na to wygląda. Podczas sumowania dwóch liiczb gdy jedna jest dużo mniejsza od drugiej to ta mniejsza nie ma żadnego wpływu. Pozostaje po prostu ta wieksza liczba. Czyli wynik będzie taki sam jak ta druga liczba.

3.Jest dany procesor o 4-bitowym słowie rozkazowym i poniższym kodowaniu rozkazów. Zapisać program w postaci mnemoników i podać wartości w rejestrach po wykonaniu 4 rozkazów: h_3,h_2,h_1,h_0. Wartości początkowe rejestrów to 0.

1) ii v 0 ld $v, %ri ri = v
2) ii j 1 add %ri, %rj rj = ri+rj

Próba rozwiązania:
Osobiście nie rozumiem w pełni polecenia, ale widziałem próby rozwiązania tego zadania w następujący sposób:

załóżmy nr indeksu 200763
h3=0h=0000b
h2=7h=0111b
h1=6h=0110b
h0=3h=0011b
(kolejne bity traktujemy jako kolejne pozycje w tych schematach iiv0, iij1)

//A jak dokładnie wyliczane są te ii? h2=7h=0111b czyli 2 starsze bity będą tym ii - 01? A potem to kolejny bit (tutaj 1) to będzie ta wartość j? I co potem z tym ostanim bitem co zostaje oraz liczbą 1 (iij '1')?
// ostatni bit, który zostaje determinuje która instrukcja ma zostać wykonana.
jak 0 na końcu: ld
jak 1: add
//Oki, dzięki, a co z tą liczbą która jest w kodzie (po v jest to 0 a po j jest to 1)?
No przecież ci napisał właśnie.
//Myślałem że chodzi o ten ostatni bit z tej liczby 7h=011'1'b
No bo właśnie chodzi. Na podstawie tego ostatniego bitu w liczbie decydujesz czy to ma być ld czy add.
//Oki, teraz rozumiem :D
Jesteś pewien, że rozumiesz?
//0101b: ii = 01 v/j = 0? instrukcja add=1?
Nom. Tylko, możliwe, że jutro będzie trochę inaczej ale już jak wiadomo o co tu chodzi to będzie łatwiej zrozumieć . No chyba, że da identyczne.
Rozkaz h3: zero na końcu czyli instrukcja1):
ii = 00, v=0, stąd rozkaz: *ld $0, %r0*
r0 = 0

rozkaz h2: jeden na końcu czyli instrukcja 2):
ii = 01, j = 1 zatem rozkaz *add %r1, %r1*
r1 = 0

rozkaz h1: zero - instrukcja 1):
ii = 01, v = 1 zatem *ld $1, %r1*
r1 = 1

Rozkaz h0: jeden - instrukcja 2):
ii=00, j=1, stąd rozkaz: *add %r0, %r1*
r0 = 0; r1 = 1

4 Jest dany fragment kodu procesu. Rand to funkcja generująca wartości od 0 do wartości w rejestrze. Ile procesów można uruchomić, jeśli rozmiar pamięci głównej to 1024 MB.

```
mov $h0 3, %eax
mov $0x100000, %ebx
mul %ebx

begin:
        mov %ebx, %eax
```

rand %eax
mov %eax,(%eax)
loop begin

Rozwiązanie:
0x100000 = 1048576 bajtów = 1024KB
Schemat działania:

1. dla indeksu na przyklad 200999 leci 93h do eax (nieistotne)
2. wrzuć 0x100000 do ebx
3. 93h * 0x100000 -> eax (nieistotne)

btw petli nie bedzie bo ecx nie ustawione (albo śmieci i random będzie)

4. przenieś ebx (0x100000) do eax (więc i tak wynik mnożenia jest nadpisywany)
5. losuj random z przedzialu <0, 0x100000> (0, 1024KB)
6. wrzuć wylosowaną liczbę do miejsca w pamięci o takim adresie jak wylosowana liczba

No i tu jest lipa bo nie można tak sobie pisać po pamięci losowo. Każdy z procesów i tak bedzie nadpisywał pamieć w przedziale <0, 1024KB>. Nie wiadomo też co znajduje się pod tymi adresami. Generalnie na żadnym systemie to raczej nie ruszy(SIGSEGV). Moim zdaniem odpowiedź to 0.

Bzdura. ~Mariusz Banach.

A więc panie Banach kontynuujmy dyskusję którą rozpoczeliśmy przy Patroniku
Ja twierdzę, że jest coś takiego jak kontrola dostępu do pamięci (i codziło mi o przypadek uruchomienia tego w systemie operacyjnym, nie na gołym żelazie, czy czymś archaicznym bez tych mechanizmów)

na x86_64 wersja c++11

```
#include <random>
#include <iostream>

using namespace std;

int main (int argv, char** argc){
	long int address;
	int *wsk;
	while(true){
		address = rand();
		wsk = (int*)address;
		*wsk = address;
		cout << "Zyje" << endl;
		system("free -m");
	}
}
```

efekt wykonania:

```
Zacate:/home/dwozniak/tmp # g++ -O0 -g3 kod.cpp -std=c++11 -o kod
Zacate:/home/dwozniak/tmp # ./kod
Naruszenie ochrony pamięci
```

ew. asm na 32 bit

```
.text
.global main
main:
        call rand
        mov %eax, (%eax)
        jmp main
```

```
Zacate:/home/dwozniak/tmp # ./kod-asm
Naruszenie ochrony pamięci
```

Dlatego twierdzę, że wątki same by się zabijały i odpalaj je w nieskończoność Pomińmy już podział pamięci na kod i dane i że system może pilnować żeby pamięć wykonywalna była read-only, gdzie w nią też możemy trafić

Druga bzdura. ~**Mariusz Banach**

Kod powyżej zaprezentowany generuje odwołania do losowych miejsc w pamięci. Tak w zasadzie, nawet nie losowych bo nie zaszła inicjalizacja ziarna kongruentnego generatora liniowego (rand() - brak funkcji srand). Także wywoływane jest odniesienie do cały czas tego samego adresu pamięci. Teraz tak, każdy proces ma swój obszar roboczy (Working Set). Pamięć procesu, opisana mapą pamięci (w linuxie znajduje się stosowny plik w /proc/pid/), opisuje, który region pamięci do czego służy i jakie ma przywileje / flagi. Domyślnie, przestrzenią wirtualną procesu - w systemach 32 bitowych, jest obszar 2^32 = 4GB, co oznacza, że wirtualnie proces ma 4GB pamięci roboczej. Odwołując się pod obszar z wylosowanego adresu z rand'a, wywoływana jest sytuacja skoku do komórki, która najpewniej nie została zaalakowana (strona pamięci, zawierająca ten konkretny bajt / adres). Innymi słowy, jeśli rand() zwróciło 0x102345, zaś strona pamięci 0x100000 nie została ZAALOKOWANA, a następuje odwołanie do tej strony w pamięci - to siłą rzeczy rzucany jest SIGSEGV (w Windowsie Memory Access Violation). Kłaniają się podstawy działania systemów operacyjnych starszaki. :-)

W przypadku kodu doktora Patronika, na skutek translacji adresów wirtualnych (u nas rand %eax zwraca między <0, 0x100000>) każdy proces ma zajęty inny zbiór ramek pamięci fizycznej. Tak więc program A z kodu Patronika może odwołać się do np. 0xabc, zaś program B do 0x12cd. Na tym polega separacja procesów, nie ma tu mowy o nadpisywaniu pamięci programu A przez program B. To są czasy systemu Win95 / DOS, nie WinNT.

Mam nadzieję, że to zakończy dyskusję. Pozdrawiam, **Mariusz Banach.**

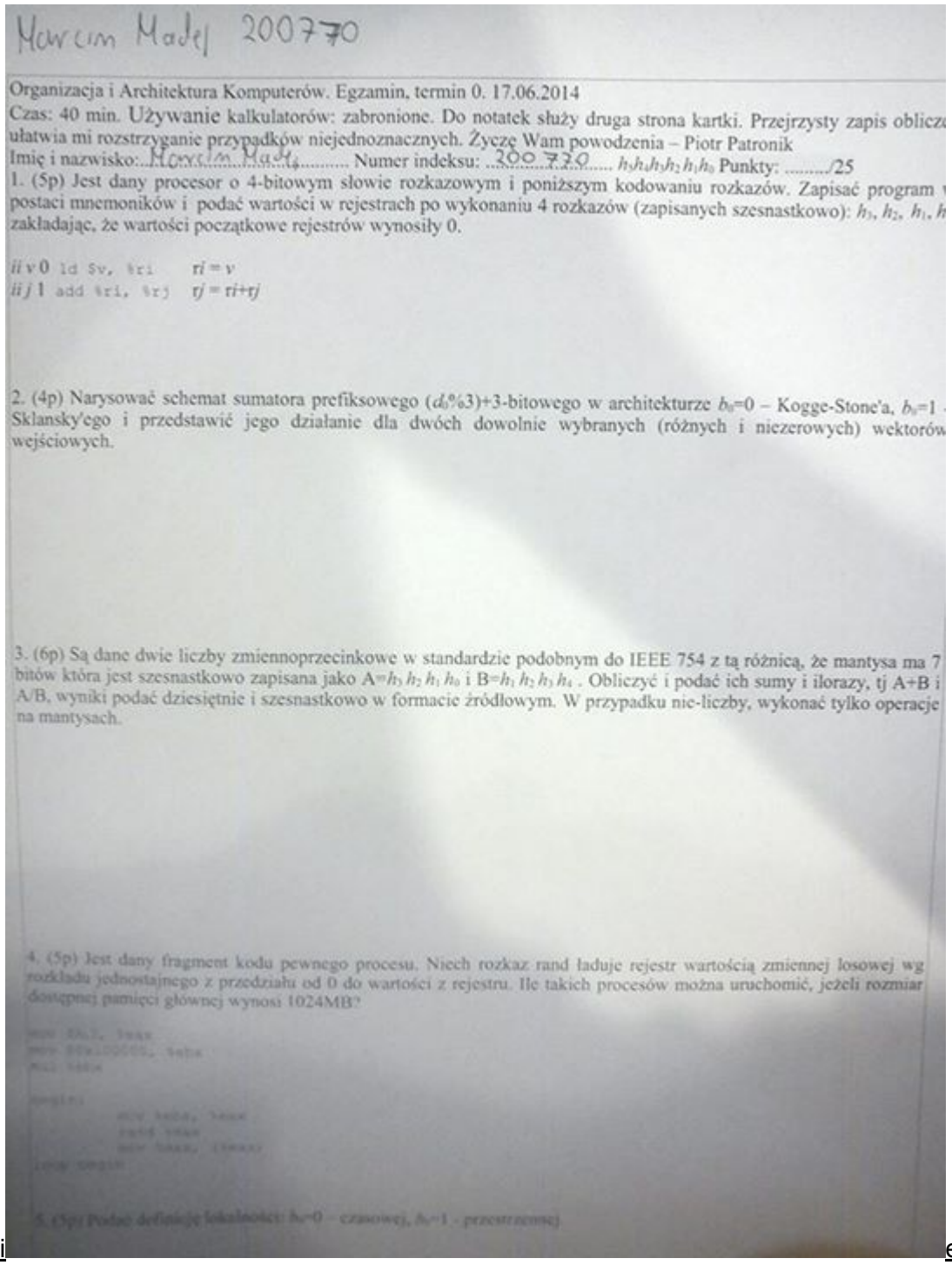
//Czyli ile takich procesów może zostać odpalone?

Odpowiedź do zadania znajdziesz w opracowaniu alternatywnym, którego link zamieściłem na początku tego dokumentu.

Odpowiedź zgodną z praktyką, udzielam: nie wiadomo ile. Jest to zależne od dostępnych zasobów pamięciowych danego systemu oraz metod jego modułu zarządzania pamięcią. W przypadku systemów z pamięcią Swap, lub plikami wymiany (pagefile.sys / Windows), lub mechanizmami ReadyBoost (posługiwanie się USB tak jakby był przedłużeniem pamięci RAM / Windows) - tam system ma większą pulę pamięci do dyspozycji w celu alokacji procesów. Innymi słowy - nowoczesne systemy operacyjne zdolne są do obsługi większej ilości procesów, niż zdolna jest pomieścić pamięć fizyczna.

5. Podaj definicję lokalności b0=0 - czasowej, b1=1 - przestrzennej

1. Lokalność czasowa oznacza tendencje do powtarzania odwołań, realizowanych w niedawnej przeszłości.
2. Lokalność przestrzenna oznacza tendencje do odwołań do obiektów umieszczonych w obszarze adresowym obejmujacym obiekty, które były juz użyte w programie.

Tutaj zdjeci  e:

Marcin Madej 200770

Organizacja i Architektura Komputerów. Egzamin, termin 0. 17.06.2014

Czas: 40 min. Używanie kalkulatorów: zabronione. Do notatek służy druga strona kartki. Przejrzysty zapis oblicz ułatwia mi rozstrzyganie przypadków niejednoznacznych. Życzę Wam powodzenia – Piotr Patronik

Imię i nazwisko: Marcin Madej Numer indeksu: 200770 $h_5h_4h_3h_2h_1h_0$ Punkty: ...../25

1. (5p) Jest dany procesor o 4-bitowym słowie rozkazowym i poniższym kodowaniu rozkazów. Zapisać program postaci mnemoników i podać wartości w rejestrach po wykonaniu 4 rozkazów (zapisanych szesnastkowo): h_3, h_2, h_1, h zakładając, że wartości początkowe rejestrów wynosiły 0.

ii v 0 `ld $v, %ri` $ri = v$
ii j 1 `add %ri, %rj` $rj = ri+rj$

2. (4p) Narysować schemat sumatora prefiksowego (d_0%3)+3-bitowego w architekturze b_0=0 – Kogge-Stone'a, b_0=1 Sklansky'ego i przedstawić jego działanie dla dwóch dowolnie wybranych (różnych i niezerowych) wektorów wejściowych.

3. (6p) Są dane dwie liczby zmiennoprzecinkowe w standardzie podobnym do IEEE 754 z tą różnicą, że mantysa ma 7 bitów która jest szesnastkowo zapisana jako A=$h_5\,h_2\,h_1\,h_0$ i B=$h_1\,h_2\,h_3\,h_4$. Obliczyć i podać ich sumy i ilorazy, tj A+B i A/B, wyniki podać dziesiętnie i szesnastkowo w formacie źródłowym. W przypadku nie-liczby, wykonać tylko operacje na mantysach.

4. (5p) Jest dany fragment kodu pewnego procesu. Niech rozkaz rand ładuje rejestr wartością zmiennej losowej wg rozkładu jednostajnego z przedziału od 0 do wartości z rejestru. Ile takich procesów można uruchomić, jeżeli rozmiar dostępnej pamięci głównej wynosi 1024MB?

```
mov $0x3, %eax
mov $0x100000, %ebx
mul %ebx

loop:
        mov %eax, %ecx
        rand %eax
        mov %eax, (%ecx)
jmp loop
```

5. (5p) Podać definicję lokalności: b_0=0 – czasowej, b_0=1 - przestrzennej