h

ujebiemy...

1 \\Parking

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

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

OIAK Kolokwium termin 0 => wyniki

https://www.facebook.com/groups/175246409273826/509672839164513/

KOŁO 2015 pytania http://tnij.org/patron2015ak

OGŁOSZENIA:

Poszukiwane rozwiązanie zadania z asemblera (v. 2015)

po prostu dodaj do doca ;)

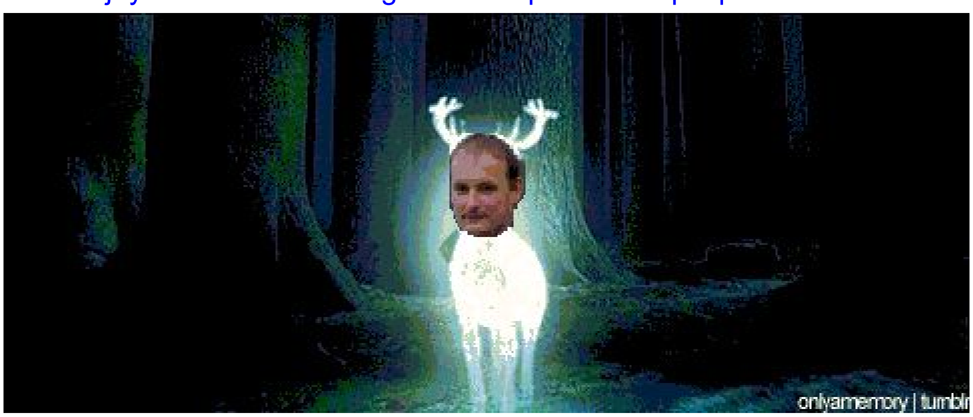
Alternatywne opracowanie zadań 1,2,4:

;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

https://docs.google.com/document/d/1ndYdjLSHIUGdbgT4wFhxPHHmA1JG9dbsv3b8U8JFjws/edit

Slajdy dra Patronika z tego roku: http://ani.cba.pl/opium/oiak-k15/



EXPECTO PATRONUS!

A co jeśli Patron to tylko projekt Architekta Biernata??..

s

h

…

Treści zadań:

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

Ostatnio były 3 grupy ma ktoś może pojęcie jak wygląda 3 rysunek?

Do obliczenia liczby poziomów: P=$\log_2 n$, gdzie n to liczba bitów, wynik podciągamy w górę, tzn. np dla n=13, p=$\log_2 13$=4

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 studenta

// %3 - modulo 3 (reszta z dzielenia przez 3)

// +3 - plus (dodac) 3

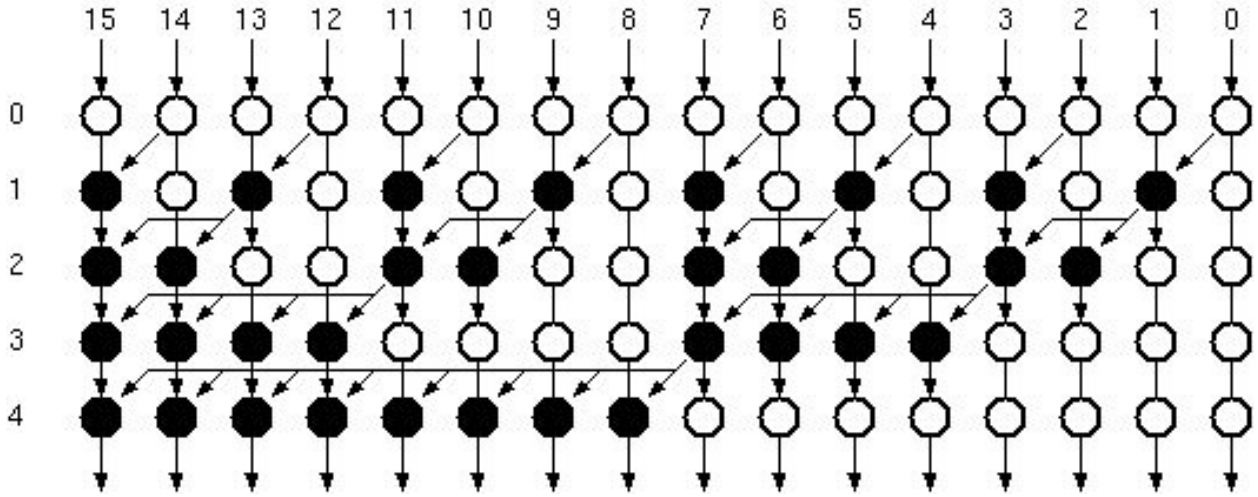
// b0 - najmniej znaczacy bit numeru indeksu studenta - b0=0 indeks parzysty, b0=1 nieparzysty

//czyli gdy indeks kończy się na np. 0 to mamy narysować sumator 3-bitowy w arch. de_Kobble'a Stone'a (WTF xD a nie koga-stona ?)(jak juz to kogga-stone'a)(jak już to kogge'a-stone'a). czyli z tego rysunku poniżej mają zostać tylko kolumny 0, 1 i 2 ?

s

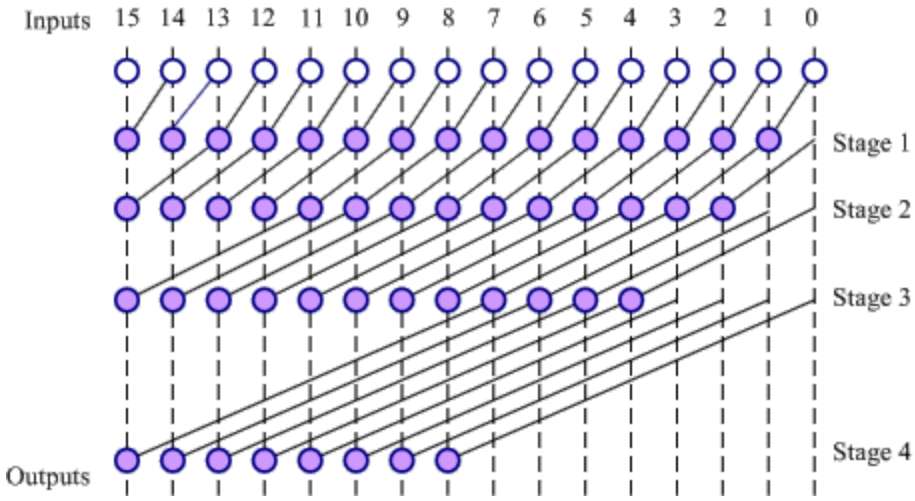
h

// chyba tak



//na pewno tak jest, że się po prostu obcina kolumny

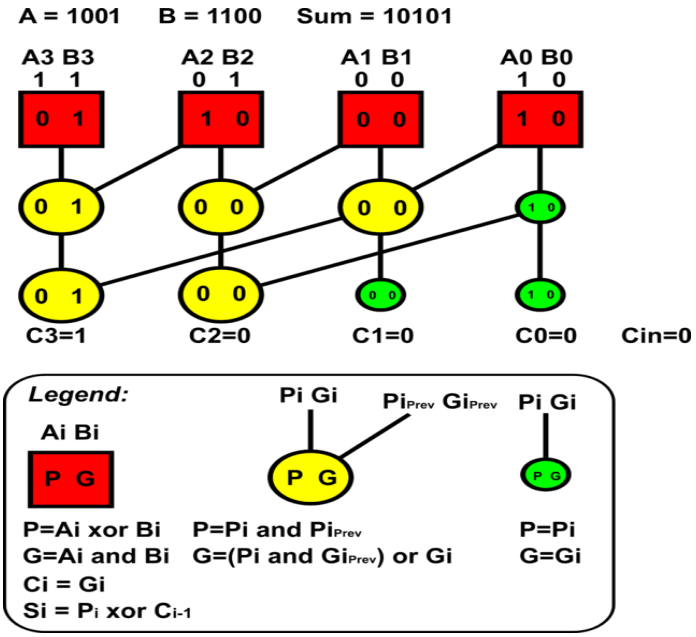
Slansky:
de_Kobble-Stone:



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

s

h



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.
Zamiast kreski najlepiej napisać np. “overflow=1” i tyle.
//gdyby ktoś się zastanawiał, to przy wzorze na sumę Pi jest P pochodzącym z czerwonego (poprzedniego - Gprev i Pprev to odpowiednio generacja i propagacja z POPRZEDNIEJ pozycji) 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.
POLICZYŁBY KTOŚ PORZĄDNIE TO DZIELENIE, BO KURWICA BIERZE HELP HELP
//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	to w ogole jest dobrze?
B:	9689	1	00101101	0001001	

s

h

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.

s

h

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:

s

h

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
// - skąd wiadomo którą wartość 'i' wybrać? mamy tu 0 i 1 dla ii=01. Chyba, że jest to po prostu liczba zapisana w binarnym i dlatego mamy r1, a nie r0?
// - 00 -> 0, 01 -> 1, 10 -> 2, 11 -> 3
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

s

h

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 $h_0$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;
```

s

h

```
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

s

h

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 ch 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.

//Czyli dalej nie ma jednoznacznej odpowiedzi na te zadanie?

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 zdjecie:

s

h

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 $h_3h_4h_3h_2h_1h_0$ 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_3 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?

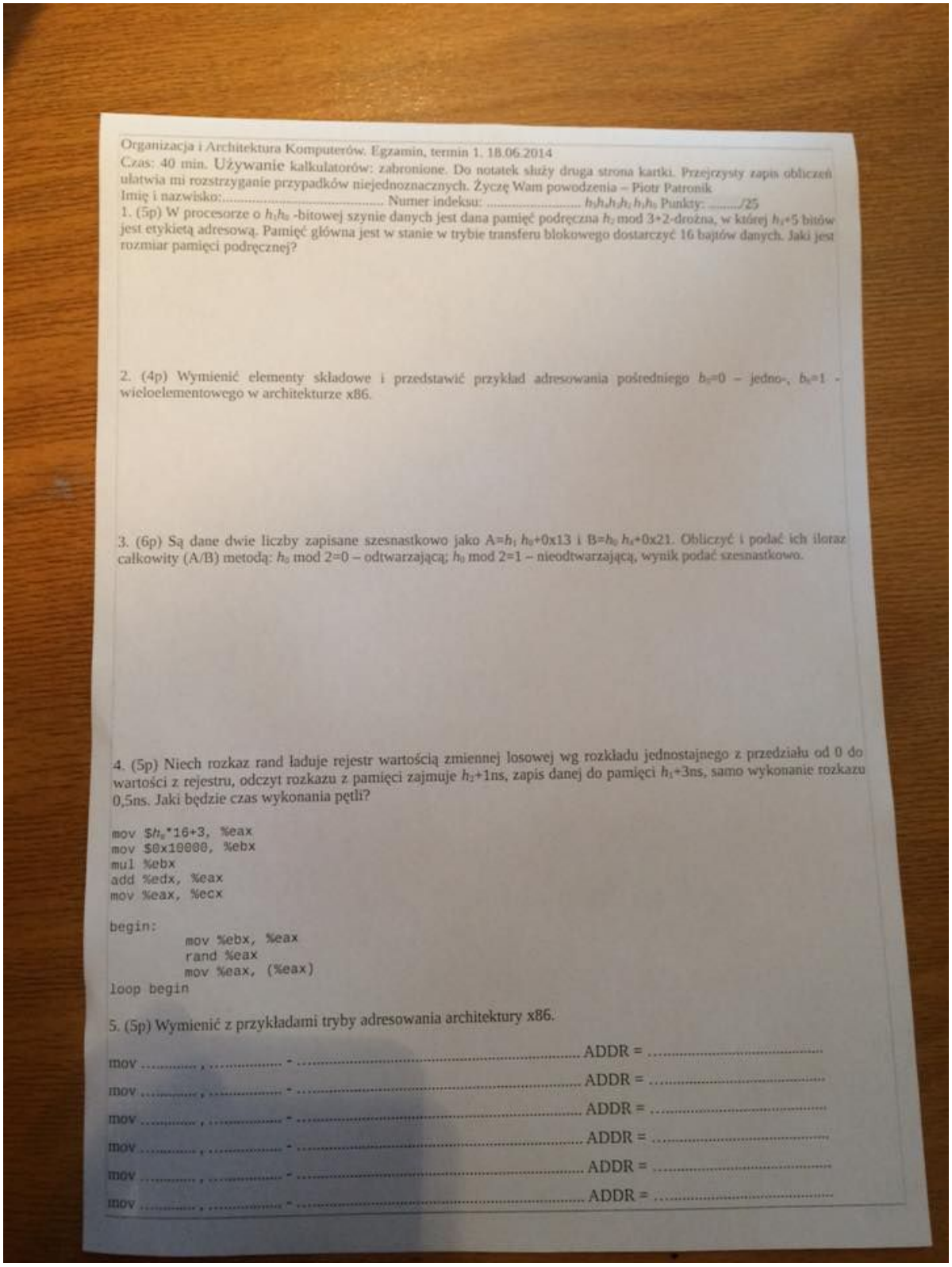
```
mov $0x1, %eax
mov $0x100000, %ebx
mul %ebx

begin:
        mov %ebx, %eax
        rand %eax
        mov %eax, (%eax)
jmp begin
```

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

s

h

KOŁO 1 TERMIN 2015



Organizacja i Architektura Komputerów. Egzamin, termin 1. 18.06.2014
Czas: 40 min. Używanie kalkulatorów: zabronione. Do notatek służy druga strona kartki. Przejrzysty zapis obliczeń ułatwia mi rozstrzyganie przypadków niejednoznacznych. Życzę Wam powodzenia – Piotr Patronik
Imię i nazwisko:..... Numer indeksu: ..... $h_3h_2h_1h_0$ Punkty: ...../25

1. (5p) W procesorze o h_1h_0 -bitowej szynie danych jest dana pamięć podręczna h_2 mod 3+2-drożna, w której h_4+5 bitów jest etykietą adresową. Pamięć główna jest w stanie w trybie transferu blokowego dostarczyć 16 bajtów danych. Jaki jest rozmiar pamięci podręcznej?

2. (4p) Wymienić elementy składowe i przedstawić przykład adresowania pośredniego h_5=0 – jedno-, h_5=1 - wieloelementowego w architekturze x86.

3. (6p) Są dane dwie liczby zapisane szesnastkowo jako A=h_1 h_0+0x13 i B=h_0 h_4+0x21. Obliczyć i podać ich iloraz całkowity (A/B) metodą: h_0 mod 2=0 – odtwarzającą; h_0 mod 2=1 – nieodtwarzającą, wynik podać szesnastkowo.

4. (5p) Niech rozkaz rand ładuje rejestr wartością zmiennej losowej wg rozkładu jednostajnego z przedziału od 0 do wartości z rejestru, odczyt rozkazu z pamięci zajmuje h_2+1ns, zapis danej do pamięci h_1+3ns, samo wykonanie rozkazu 0,5ns. Jaki będzie czas wykonania pętli?

```
mov $h0*16+3, %eax
mov $0x10000, %ebx
mul %ebx
add %edx, %eax
mov %eax, %ecx

begin:
        mov %ebx, %eax
        rand %eax
        mov %eax, (%eax)
loop begin
```

5. (5p) Wymienić z przykładami tryby adresowania architektury x86.

mov ..... , ... - ..... ADDR = ...
mov ..... , ... - ..... ADDR = ...
mov ..... , ... - ..... ADDR = ...
mov ..... , ... - ..... ADDR = ...
mov ..... , ... - ..... ADDR = ...
mov ..... , ... - ..... ADDR = ...

s

h

Zad. 1
Co patron kazał dopisać w tym zadaniu? Z tego co pamiętam kazał zmienić coś w szynie bitowej. Nie miało to być +32 ?
// Miało być 32 + h1
To zadanie jest bardzo podobne do zadań które dawał biernat na swoim kole.
https://docs.google.com/document/d/1ruzpUn0eqahelJ-hvL0By1m7mYF16ZyiEb06DIs5nxU/edit# strona ~~ 16 . Jednak nie bardzo wiem jak to poskładać - my mamy mniej danych

Rozwiązane na podstawie:

- http://www.d.umn.edu/~gshute/arch/cache-addressing.xhtm l -> An example
- http://www.eecg.toronto.edu/~moshovos/ECE243-07/l26-caches.html -> Example #2

TAG adres lini	SET INDEX indeks linii	BLOCK OFFSET indeks słowa

Postać adresu w RAM wg http://www.d.umn.edu/~gshute/arch/cache-addressing.xhtml (nazwy po polsku wg slajdów JB)

Szerokość adresu = rozmiar szyny.

Niech szyna ma 38 bitów, cache jest 4-drożne a liczba bitów etykiety wynosi 14. Wiemy, że w trybie blokowym pamięć przesyła $2^4 = 16$ (adresacja jest bajtowa) bajtów, czyli rozmiar bloku(BLOCK OFFSET) to 4.
Adres linii (TAG)(czyli rozmiar etykiety adresu) to 14 bitów .
Zostaje 20 na SET INDEX.

Nawiązując do linku:

$$liczba\ setów\ cache\ = \frac{Rozmiar\ cache}{Ilosc\ bajtów\ w\ secie}$$

+`Znając ilość bajtów w secie (4 drogi * 16 bajtów = 64) oraz ilość setów (maksymalna wartość set index) 2^{20} możemy wyliczyć rozmiar cache jako iloczyn ilości setów cache i ilości bajtów w wierszu. Mamy zatem:

$$2^{20} * 64\ \sim 64\ MB$$

PS. Niech ktoś to zweryfikuje, wytknie błędy

~Michał Pawlik

//// No spoko, ale w moim nowym lapku jest 256 KB pamięci cache :D 64mb cache jest w dyskach twardych
/// Z moimi danymi (40-bitowa szyna, 2-drożna pamięć i 5-bitów etykieta wychodzi mi 65536 MB także chyba sposób jest zły :|
//przeanalizowałem to i wyszło mi 32gb cache, a wydaje się poprawnie
//Jest to raczej błędne rozumowanie - wartość cache jest dość mała !
// Nie twierdzę, że to jest zupełnie dobre, podrzuciłem pomysł do dyskusji, bo lepszy taki niż żaden

s

h

// Ktoś zaproponuje inne wnioski z linków powyżej rozwiązania?
//Wydaje się dobrze^

Zad. 2 - rozwinąć/zweryfikować, ja to wypisałem z książki Biernata
Pośrednie jednoelementowe -
Elementy:

- Według mnie elementy to operand i adres docelowy //Zgadzam się

Przykłady:

- bezwzględne
- bezwzględne pośrednie
- rejestrowe bezpośrednie
- rejestrowe pośrednie
- rejestrowe pośrednie z modyfikacją

Pośrednie wieloelementowe -
Elementy:

- baza (adres odniesienia w rejestrze procesora)
- przemieszczenie bazy (stała adresowa, w adresowaniu dwupoziomowym dodawana do bazy, stanowiąca osobne słowo rozszerzenia kodu rozkazu)
- indeks (umieszczony w rejestrze procesora wraz z mnożnikiem skali wskazanym w słowie lub słowach rozszerzenia kodu)
- relokacja (stała dodawana do wyznaczonego adresu, stanowiąca osobne słowo rozszerzenia kodu rozkazu)

Przykłady (dwuelementowe):

- bazowe z przemieszczeniem
- indeksowe z przemieszczeniem
- bazowo-indeksowe
- względne z przemieszczeniem
- względne indeksowe

//składowe dla jedno i wieloelementowego adresowania są takie same, czy różne? Bo w zadaniu jest, by podać dla jedno lub wielo, ale na slajdach nie ma wzmianki o elementach składowych adresowania jednoelementowego(a najlepiej jak ktoś wiedzący podałby przykład).

Zad. 3
Dzielenie nieodtwarzające w systemie uzupełnieniowym (ogólne przypomnienie)
http://www.pwrhelp.za.pl/ak1/dzielenieUzup_v11.pdf
Ma ktoś jakies materiały do przypomnienia odtwarzającego?

s

h

http://sendfile.es/pokaz/399860---pB4b.html
Ile tam jest na fotce w B? H0h1 czy h0h4 czy h0hA ?? Pyt. 2016
PYTANIE:
O co chodzi z zapisem tej liczby w formacie h1h0+0x21. Jeżeli np h1=4 a h0=7, to h1h0 to liczba 47 (dziesiętnie), którą trzeba przekonwertować na system 16-tkowy, a następnie dodać do 0x021, żeby uzyskać liczbę końcową?

//A to nie jest tak ze 47 to juz jest w hex ? i dodajemy do tego 0x21. To bedzie 68(hex) tak ?
//Tak, 47 jest już w hex, dodajemy 21 i mamy 68. Zawsze można też zapytać Patronika na kole. tez tak myślę

Może chodzi o to, że gdy h1=4 i h0=7:
A=h1 h0+0x13 to tylko do h0 dodajemy 0x13?

//Nie , ja na kole zrobilem tylko to zadanie + jakies adresowanie i mialem 10 pkt wiec bedzie tak jak wyzej pisza // w jaki sposób je rozwiązałeś? przekształciłeś na NB/U2 ?
p//z hex na binarke, rozszerzenie (0) i dziele obie
PROSZĘ O SPRAWDZENIE poniżeszego rachunku, coś mi się nie zgadza, zakładając, że zamieniamy liczbę z hex na u2 to mamy liczbę ujemną przez dodatnią i wynik powinien wyjść ujemny, w takim razie czo jest nie tak z pierwszym działaniem ?

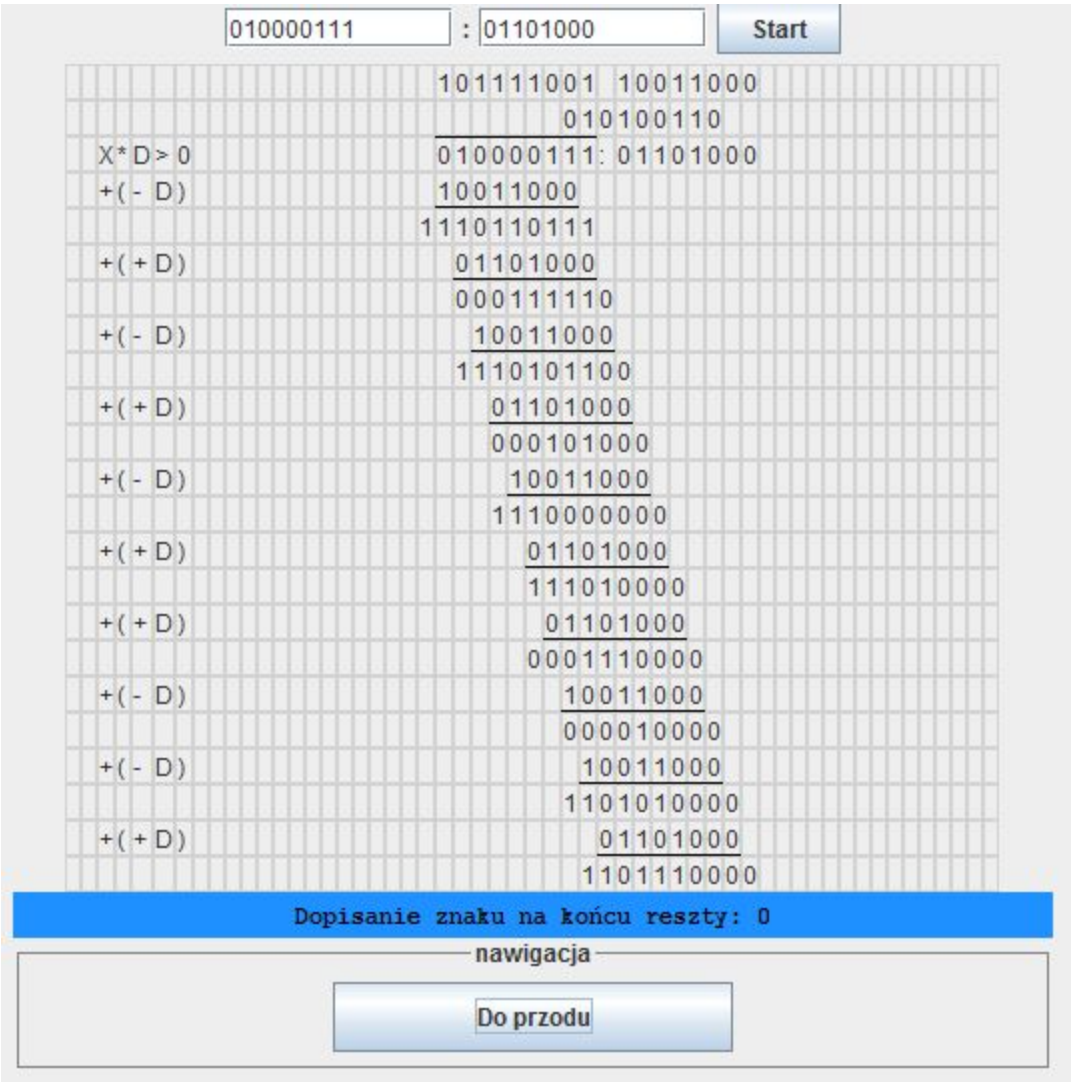
```
87 : 68   // czy 87 to liczba ujemna ????? - NIE!
 Dzielenie jest źle, skoro 87 jest liczbą dodatnią to powinna wyglądać tak:
(0)10000111 O ile te liczby tak właśnie wyglądają // racja

(1)0000111  : (0)1101000          znaki niezgodne +
 +01101000
  111011110            niezgodne +   q0=0
+  01101000
   010001100                zgodne -    q1=1
-   01101000
     01001000               zgodne -    q2=1
-    01101000
      11000000          niezgodne +      q3=0
+     01101000
       01010000             zgodne -    q4=1
        01101000
        11010000        niezgodne -      q5=0
-       01101000
5        01110000       zgodne -    q6=1
-     __ 01101000
          0001000       zgodne -    q7=1
```

s

h

```
Wynik 01101011    = 6B        // to dzielenie jest poprawne, ale to nie 87 i
68
```

Istnieje kalkulator obliczający dzielenie w u2:



07:32
Kalkulator: http://speedy.sh/tJ7s3/dzielenie-nieodtwarzajace.jar / dzięki!

s

h

Zad. 4
Czy wartości znajdujące się w rejestrach mają tu jakiekolwiek znaczenie? W pętli mamy dwie operacje mov i jedno wywołanie rozkazu rand, nie wystarczy dodać do siebie 2x zapis danej do pamięci + odczyt rozkazu z pamięci + wykonanie rozkazu? Chyba niezależnie od wartości w rejestrze eax pętla będzie się wykonywała tak samo.
// a eax nie jest wpisywany czasami do ecx z ktorego korzysta loop ? np u mnie wychodziło w ecx 30000 wiec petla wyknoywala sie 30000 razy, ale nie wiem dokładnie czy tak działa loop.
// Pytanie co rozumiemy przez czas wykonania się pętli - czas jednokrotnego wykonania się pętli, czy czas wykonania wszystkich powtórzeń? ;] Według mnie nie jest to jednoznaczne.
// Na kole napisałem tylko jeden obieg pętli, a także ile znajduje się w rejestrze ecx i eax, zobaczymy jak to oceni pepe, bo rzeczywiscie nie było napisane czy całkowity czas wykonania czy np jeden obieg// i jak to ocenił?
// Ktoś go pytał czy chodzi o jednokrotne wykonanie, czy czas wszystkich powtórzeń. Odpowiedział, że czas wszystkich powtórzeń.
//Szkoda że nie ma w zwyczaju ogłosić tego wszystkim piszącym egzamin gdy padnie takie pytanie.

W zadaniu ważne jest poprawne określenie gdzie zaczyna się pętla i gdzie kończy. W skład jednej iteracji pętli wchodzą w tym przypadku cztery instrukcje:

1. mov %ebx, %eax
2. rand %eax
3. mov %eax, (%eax)
4. loop Begin

Po wyliczeniu czasów konkretnych operacji można łatwo obliczyć czas wykonania jednej iteracji pętli; ja zakładam, że: odczyt rozkazu = 9ns, zapis do rejestru (Zapis danej do pamięci????)= 8ns, wykonanie rozkazu 0,5ns.

1. 9+0,5 = 9,5ns
2. 9+0,5 = 9,5ns
3. 9+8+0,5 = 17,5ns
4. 9+0,5 = 9,5ns
RAZEM: 46ns

Kolejnym etapem jest określenie ilości iteracji. W instrukcji 1. do %eax ląduje wartość w zależności od indeksu (u mnie $83); w 2. Zamieniamy z hex na dec żeby się lepiej liczyło - $0x10000_{(HEX)} = 2^{16} = 65536_{(DEC)}$. Następnie mnożymy wartości i otrzymujemy 5 439 $488_{(DEC)}$ i ta liczba mieści się w rejestrze 32-bit, więc trafi w całości do rejestru %eax (wynik mnożenia %edx – starsza część wyniku, %eax – młodsza). Po wykonaniu 3. instrukcji w %eax będziemy mieli tę

s

h

samą wartość (5 439 488). Instrukcja 4. skopiuje wartość w %eax do %ecx (to będzie ilość iteracji pętli).

Mamy już wszystko co potrzeba do obliczenia czasu wykonywania pętli. Należy również wiedzieć jak działa instrukcja loop. Otóż robi ona cmp $0, %ecx i jeśli wartość jest większa od 0 to wykonuje skok do etykiety (tutaj begin).

Czas wykonywania petli wynosi zatem: **czas iteracji * ilość iteracji**,
czyli **46*5439488 = 250 216 448ns**.

//Dlaczego operację : **mov %ebx, %eax** nie traktujesz jako: odczyt+ **zapis do rejestru** + wyk. rozkazu?
//Mariusz(nie Banach!): Bo zapis to zapis do rejestru a nie do pamięci (tak mi sie wydaje :P)
// Patryk : loop dekrementuje ecx wiec chyba też powinien być odczyt i zapis, ale nie jestem pewien. Rand według patronika "ładuje" więc chyba sam zapis.
//Mariusz: Pytałem patronika na kole i mówił, że rozpatrujemy to na poziomie kodu.
// Patryk: czyli według niego jeśli nie mamy konkretnego zapisu w kodzie to nie bierzemy pod uwagę. Heh bez sensu, bo instrukcje sobie a on sobie :D
//Mariusz: TAK! To jest gość z ZAKa. Jego mózg pracuje na "Higher level of abstraction" :D
//Piotrek: czyli fakt, że przy instrukcji mov %eax, (%eax) prawdopowodnie program się wykrzaczy Bo będziemy próbowali zapisać gdzieś w pamięci, gdzie może nam nie wolno, nie ma znaczenia? Chyba mi mignęła już jakaś dyskusja na ten temat, ale nie moge odnaleźć.

zad. 5.

		//~**O co chodzi z ADDR?**~
MOV $5, %eax	-adresowanie bezpośrednie	ADDR= %eax
MOV %eax, %ebx	-adresowanie rejestrowe	ADDR= %ebx
MOV (0x4321), %eax	-adresowanie pośrednie	ADDR=
MOV (%esi), %eax	-adresowanie rejestrowe pośrednie	ADDR= %ebx
MOV (%ebx, %edi), %eax	- adresowanie bazowo-indeksowe	ADDR=

MOV %eax, przemieszczenie(%ebx)
- adresowanie bazowe z przemieszczeniem ADDR = przemieszczenie + ebx

http://staff.ustc.edu.cn/~xlanchen/cailiao/x86%20Assembly%20Programming.htm - w tym linku jest akurat 6 i są wzory na wyliczanie adresów, myślę, że dokładnie o to chodziło.
GDZIE W TYM LINKU SA NIBY WYLICZANE TE ADRESY?

http://download-mirror.savannah.gnu.org/releases/pgubook/ProgrammingGroundUp-1-0-booksize.pdf 41 strona. Według tego co tu jest to te adresy się inaczej nazywają. Moim zdaniem w 5 od góry jest błąd powinno być tak: MOV (%ebx, %edi,1), %eax

s

h

// Czy rejestrowe pośrednie z modyfikacją to nie jest np pop/push? Adres zawarty jest w ESP, który jest modyfikowany podczas wykonania
Jak to wybrałem z tego 6 które są na pewno dobrze(chyba) i po prostu je wpiszę jbc: natychmiastowe, bezwzględne, rejestrowe bezpośrednie, bazowe z przemieszczeniem, indeksowe z przemieszczeniem i bazowo-indeksowe

07:32

ZROBIŁ KTOŚ ZDJĘCIE KOŁA 2 ?

Szamotanie - Proces się szamoce jesli spędza więcej czasu na stronicowaniu niż na wykonaniu.

Biernat:
Szamotanie następuje gdy:
Suma zbiorów roboczych procesów aktywnych > rozmiar dostępnej pamięci
Heurystyka:
Nie wymieniaj bloku, który jest częścią zbioru roboczego aktywnego procesu i nie uaktywniaj procesu, którego zbiór roboczy nie może zostać w całości odwzorowany w pamięci głównej
.

Warunki wstępne -
System dąży do utrzymania wykorzystania procesora ˙ – Jesli użycie procesora spada – zwiększany jest stopień wieloprogramowości. Uzywany jest algorytm globalnego zastępowania stron. Jesli proces potrzebuje więcej ramek pamięci – może je odebrać innym procesom ´
Mechanizm powstawania szamotania
• Jeden z procesów zwiąksza zapotrzebowanie na ramki pamięci: – Ramki są zabierane innym procesom – Zblizając się do minimalnej ilości ramek, procesy częściej wykonują stronicowanie ´

s

h

• Wzrost zapotrzebowania na urządzenie stronicujące: – Procesy oczekują w kolejce do pamięci pomocniczej – Spadek wykorzystania procesora Mechanizm powstawania szamotania • System obserwuje spadek wykorzystania procesora: – Następuje zwiększenie stopnia wieloprogramowosci. . . ´
• Nowe procesy potrzebują ramek pamięci: – Dalsze ograniczanie ilosci dostępnych ramek ´ – Dalsze nasilenie stronicowania • System utyka – procesy cały czas sp¸edzaj ¸a na stronicowaniu Mechanizm powstawania szamotania 12

Dobrze wyjaśnione stronicowanie - http://students.mimuw.edu.pl/SO/LabLinux/PAMIEC/PODTEMAT_7/opis.html

Jak się liczy czas w sekwencji lub w potoku?

s