

# Architektury systemów komputerowych

## Lista MIPS2

$x_9 = 5$  (minimum na bdb)

Zadanie polega na napisaniu programów w asemblerze procesora MIPS. Wszystkie programy powinny wykonywać się na symulatorze MARS. Niektóre zadania wymagają użycia pewnych narzędzi dostarczonych z MARSem (zakładka *Tools*). Każde z narzędzi ma pomoc (przycisk *Help*), która wyjaśnia jak go używać. W niektórych zadaniach będzie trzeba użyć pewnych funkcji systemowych, które nie były omówione na wykładzie (są one opisane w pomocy programu MARS – zakładka *MIPS*, podzakładka *Syscalls*). Oprócz samego programu należy przygotować także dane, na których będzie można zaprezentować jego działanie. Zadania należy przygotować na zajęcia w tygodniu 5-9 maja.

Zasady oddawania zadań w moich grupach: najpóźniej w dniu poprzedzającym ćwiczenia, do godz. 22.00, należy przysłać mi mailem (na adres [kiero@cs.uni.wroc.pl](mailto:kiero@cs.uni.wroc.pl), temat wiadomości: 'MIPS') skompresowany programem zip plik o nazwie 'INa.zip' (gdzie 'I' to pierwsza litera imienia, a 'Na' dwie pierwsze litery nazwiska autora), zawierający pliki o nazwach 'INa.asm' (gdzie 'INa' jest jak wyżej, a 'i' to numer zadania) z rozwiązaniami poszczególnych zadań. Na ćwiczenia (które odbędą się najprawdopodobniej w sali 141) należy przyjść ze standardową deklaracją z zaznaczonymi numerami przysłanych zadań. Wybrane osoby będą prezentowały swoje rozwiązania zadań (z mojego laptopa, na ekranie telewizora lub przy pomocy rzutnika). Podczas prezentacji należy przedstawić działanie programu, zaprezentować, omówić jego kod i odpowiedzieć na ewentualne pytania prowadzącego lub innych studentów (przed ćwiczeniami należy dobrze przypomnieć sobie swoje rozwiązania; odpowiedź typu: 'pisałem to parę dni temu i już nie pamiętam co robi ten fragment kodu', czy 'nie wiem jak działa ten rozkaz' kwalifikuje się na „grzybka”).

1. Napisz program symulujący jednowymiarową grę w życie (wariant znanej gry Conway'a). Pojedyncze pokolenie opisujemy jako ciąg zer i jedynek długości 32. Zero oznacza komórkę martwą, jedynka – żywą. W następnym pokoleniu zmiany są następujące: komórka żywa, która ma 0,1 lub 3 sąsiadów umiera, komórka martwa, która ma 2 lub 3 sąsiadów ożywa. Sąsiadami komórki są komórki oddalone od niej o 1 lub 2 pozycje. Napisz procedurę, która interpretuje parametr otrzymany w rejestrze \$a0 jako opis jednego pokolenia i zwraca w rejestrze \$v0 opis kolejnego pokolenia. Program główny powinien wczytywać opis pierwszego pokolenia, a następnie cyklicznie generować i wypisywać kolejne pokolenia. Kolejne pokolenia wypisuj np. używając kropek na oznaczenie komórek martwych, a gwiazdek na oznaczenie komórek żywych.
2. Używając rozkazów zmiennopozycyjnych napisz program wyznaczający iteracyjnie pierwiastek kwadratowy danej liczby za pomocą następującego wzoru.

$$x_{n+1} = \frac{x_n + a/x_n}{2}.$$

Dokładność obliczeń zadana jest parametrem  $\epsilon$ . Obliczenia należy zakończyć, gdy  $|x_{n+1} - x_n| < \epsilon$ . Program powinien wczytywać  $a$  i  $\epsilon$  z klawiatury, a wynik wyświetlić na ekranie.

3. Napisz program zliczający niepuste linie w pliku, do którego ścieżka przekazana jest w rejestrze \$a0. Linie (za wyjątkiem ostatniej) kończą się znakiem LF (line feed). Za pustą linię uznajemy taką, która składa się z zera lub więcej spacji. Wynik umieść w rejestrze \$v0.
4. Pewne urządzenia mogą posiadać własną pamięć widoczną dla procesora. Używając narzędzia *Bitmap Display* napisz program, który w pętli losuje różnokolorowe prostokąty (odpowiedni *syscall*) i rysowuje je (boki prostokąta oraz jego „wnętrze”) do pamięci „ekranu”. Należy uśrednić kolor (tj. składowe RGB) rysowanego prostokąta z kolorem tła. (Dokładniej: losowane są współrzędne prostokąta oraz składowe RGB koloru. Ostateczna wartość każdej składowej koloru pojedynczego punktu  $P$  rysowanego prostokąta ma być średnią wartości wylosowanej oraz aktualnej wartości tej składowej w  $P$ .)
5. W tym zadaniu zobaczymy jak może wyglądać komunikacja z prostymi urządzeniami wejścia-wyjścia. Używając narzędzia *Digital Sim Lab* napisz grę typu „ciepło – zimno”. Będzie ona

polegać na tym, że komputer wylosuje pewną liczbę z zakresu 0 – 255. Zadaniem użytkownika będzie w ciągu 8 tur znaleźć wylosowaną liczbę. Gdy wpisana liczba jest zbyt mała na wyświetlaczu ma się pojawić napis “lo”, jeśli za duża to “hi”, a program powinien czekać na następną liczbę. Jeśli podana liczba jest równa lub przekroczona dozwoloną ilość kroków to przez kilka sekund mają mrugać kropki i gra ma się zrestartować.

6. (2 pkt.) Procesory MIPS nie potrafią wczytywać (zapisywać) słów znajdujących się pod adresami niepodzielnymi przez rozmiar słowa – przy próbie takiej operacji wygenerowany zostanie wyjątek. Napisz procedurę obsługi wyjątku, która pozwoli odczytać dane spod wskazanego adresu (zapisać dane pod wskazany adres) rozbijając instrukcje odczytu (zapisu) na bajty i używając instrukcji LB, SB (nie wolno korzystać z pseudoinstrukcji ULW, USW i podobnych). Wracając z przerwania przeskocz o jedną instrukcję do przodu, tak żeby stworzyć złudzenie, że błąd nie wystąpił, a wykonanie instrukcji się powiodło. Pokaż, że Twoje rozwiązanie działa dla instrukcji LW i SW dla dowolnych adresów. Więcej na temat wyjątków można przeczytać w zakładce pomocy MARSa *MIPS, Exceptions*.
7. (3 pkt.) Napisz kompilator języka BF. \* Kod źródłowy programu powinien zostać wczytany z podanego przez użytkownika pliku. Zarezerwuj miejsce dla tablicy, do której będziesz wpisywać kolejne instrukcje procesora MIPS odpowiadające instrukcjom języka BF. Po zakończonej kompilacji uruchom kod wskazując na początek tej tablicy. Wygenerowany kod ma być samodzielny – tj. nie może korzystać z funkcji dostępnych w kompilatorze ani do niego powrócić. Rozwiązanie tego zadania w postaci interpretera jest niedopuszczalne. Przetestuj swoje rozwiązanie na kodach źródłowych BF dostępnych w sieci. Uwagi: instrukcje należy wpisywać do sekcji danych (.data) lub sterty (.heap), ale szablony tłumaczeń instrukcji BF wygodnie jest trzymać w sekcji programu (.text); narzędzie pracy krokowej nie będzie pokazywało wykonywania instrukcji w sekcji danych, ale będzie widać efekty działania tych instrukcji (np. zmiany zawartości rejestrów); być może w symulatorze MARS należy włączyć opcję “Self-modifying code”.
8. Procesory MIPS32v2 posiadają przydatne instrukcje SEB (sign extend byte), EXT (extract bits), INS (insert bits), które nie występują w MIPS32v1. Semantykę wymienionych instrukcji można znaleźć MIPS32 Instruction Set Quick Reference<sup>†</sup>. Bez użycia pseudoinstrukcji i instrukcji skoków zaimplementuj wyżej wymienione operacje z użyciem makr assemblerowych. Więcej na temat makr można przeczytać w zakładce pomocy MARSa *MIPS, Macros*. Pamiętaj, że makra nie mogą modyfikować żadnych rejestrów poza docelowym i \$at oraz powinny unikać dostępu do pamięci.

---

\*<http://pl.wikipedia.org/wiki/Brainfuck>

<sup>†</sup><http://www.cs.iastate.edu/~cs321/MD00565-2B-MIPS32-QRC-01.01.pdf>