Politechnika Wrocławska

Architektura Systemów Komputerowych

Wykład 9

Dr inż. Radosław Michalski
Katedra Inteligencji Obliczeniowej, Wydział Informatyki i Zarządzania
Politechnika Wrocławska

Wersja 1.1, wiosna 2018

Źródła i licencja

Najbardziej aktualna wersja tego wykładu znajduje się tu:
https://github.com/rmhere/lecture-comp-arch-org

Zawartość tego wykładu

Stos

Liczby zmiennoprzecinkowe

Stos

Funkcje

Jak działają funkcje?

- zwykle funkcje otrzymują pewne wejście (argumenty) i generują wyjście (zwracana wartość)
- w trakcie wywołania funkcji następuje ewaluacja argumentów
- w przebiegu programu skaczemy do funkcji i ją wykonujemy
- po klauzuli `return` następuje powrót do dalszego wykonania programu

Stos

Funkcje - rozważania

- funkcje także mogą deklarować własne zmienne (i tym samym mogą wymagać dodatkowej pamięci)
- w przypadku rekurencji nie mogą one wykorzystywać tego samego jej obszaru
- w jaki sposób działają funkcje w MIPS?

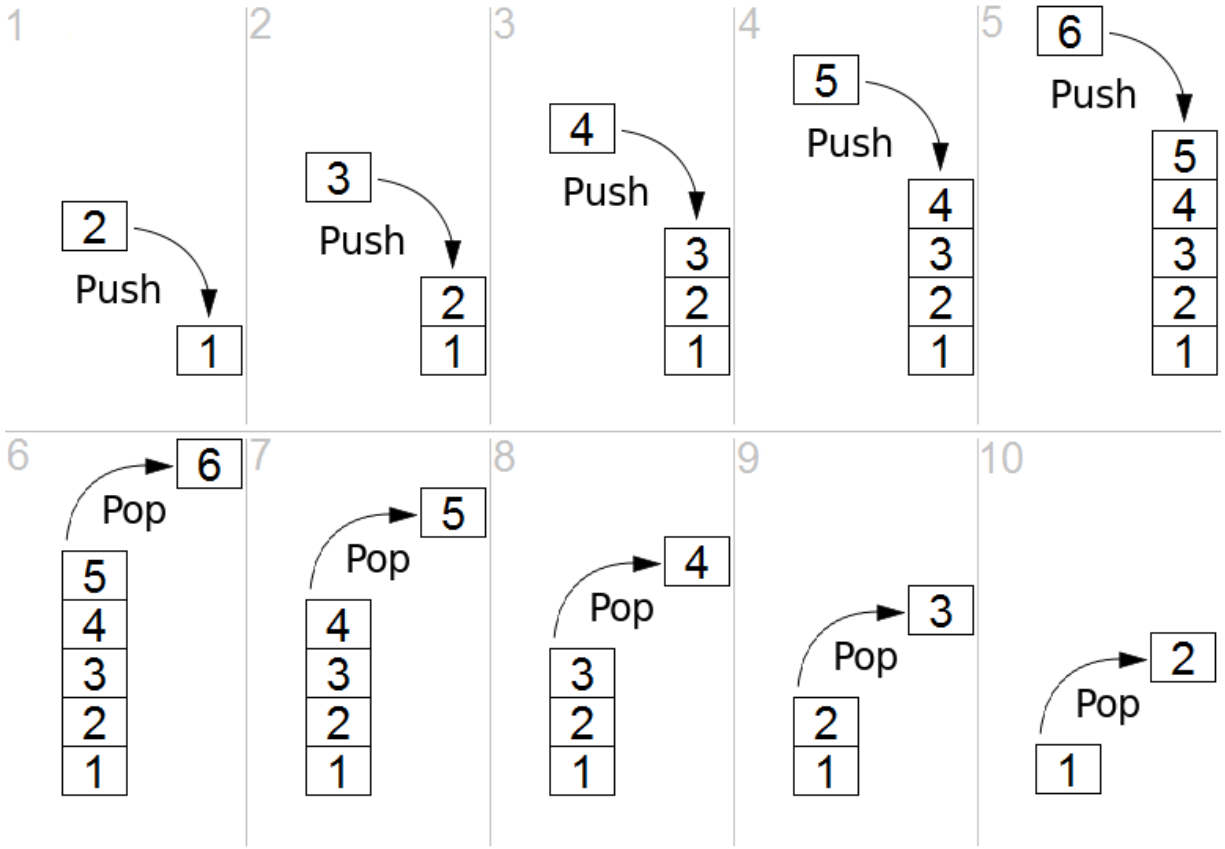
Stos

Stos - wprowadzenie

- stos - ciągły obszar pamięci
- zawiera:
  - stack limit/origin (najniższy poprawny adres stosu)
  - stack pointer (wskaźnik stosu)
  - stack bottom (najwyższy poprawny adres stosu)
- przepełnienie stosu (ang. *stack overflow)* oznacza, że *stack pointer* $<$ *stack limit*

Stos

Stos - operacje



Maxtremus, *domena publiczna*

Stos

Stos w MIPS - szczegóły

- wskaźnk stosu to rejestr \$29 (`$sp`)
- nie musisz używać \$29 jako SP, to tylko konwencja
- utrzymuj `$sp` (lub inny rejestr) ustawiony na początek "dobrych" danych w stosie

Stos

Stos - przykład

```
addi $t3, $zero, 9
push: addi $sp, $sp, -4 # Obniż SP o słowo
       sw $t3, 0($sp) # Zapisz $t3 na stosie
pop:  lw $t4, 0($sp) # Załaduj wartość do $t4
       addi $sp, $sp, 4 # Zwiększ SP o słowo
```

Stos

Stos - przykład - wiele danych

Push:

- zmniejsz $sp jednorazowo
- zapisz wiele wartości (*base addressing* względem $sp)

Pop:

- odczytaj wiele wartości (*base addressing* względem $sp)
- zwiększ $sp jednorazowo

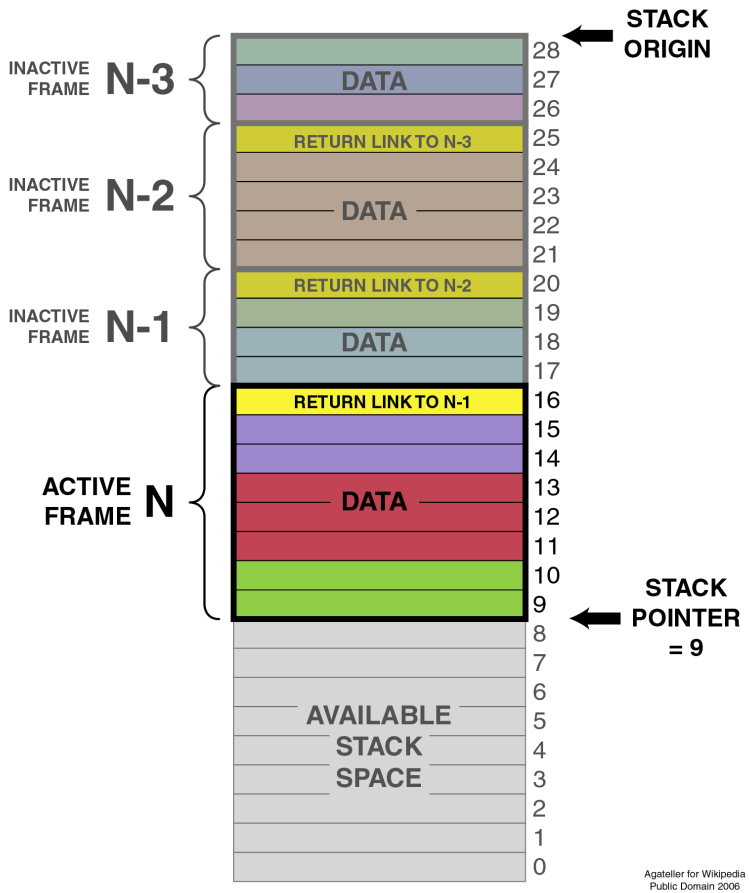
Stos

Stos i funkcje

- musimy wiedzieć jak duży obszar jest potrzebny na wywołania funkcji
  - argumenty
  - zwracana wartość
- powyższe obszary nazywane są **stack frames**
- kolejny wskaźnik - **frame pointer** (`$fp`)
- przechowuje on wartość $sp zanim zmieniono jego wartość

Stos

Ramki stosu



Agateller, *public domain*

Stos

Funkcje i rejestry

Wywołujący (caller) i wywoływany (callee):

- *caller* wywołuje *callee*
- callee nie wie kto go wywołał

Rozważania dla MIPS:

- skończona liczba rejestrów
- callee wykorzystuje *saved registers* - konwencja (x8)
- caller wykorzystuje *argument registers* - konwencja (x4)
- wartości zwracane w $v0 i $v1
- callee także może być caller - co wtedy?

Stos

Źródła i polecane materiały

- M. Hill, The MIPS Register Usage Conventions, University of Wisconsin-Madison, WI, United States (materiały uzupełniające do kurs)

Liczby zmiennoprzecinkowe

Liczby całkowite

W jaki sposób pracować z liczbami całkowitymi w systemach komputerowych?

- przykładowa liczba całkowita: 1283093714 (31 bitów)
- liczby całkowite - reprezentowane precyzyjnie
- maksymalna długość - definiowana przez architekturę
- 2^n, gdzie n oznacza liczbę bitów
- signed/unsigned
- przekroczenie zakresu

Liczby zmiennoprzecinkowe

Liczby rzeczywiste - wprowadzenie

W jaki sposób pracować z liczbami rzeczywistymi w systemach komputerowych?

- przykładowa liczba rzeczywista: 3.82379102
- nie ma możliwości dokładnej reprezentacji niektórych liczb rzeczywistych
- rejestry mają stałą długość (32 bity w przypadku MIPS)
- reprezentacja dokładna / przybliżenie
- w jaki sposób wykorzystać te 32 bity efektywnie?
- stała pozycja / zmienna pozycja (kropki/przecinka)

Liczby zmiennoprzecinkowe

Liczby rzeczywiste - stała pozycja

czesc calkowita . ulamek

3.82379102

00000011 (8 bitów) . 10011101001000000101011110 (28 bitów)

Liczby zmiennoprzecinkowe

Liczby rzeczywiste - zmienna pozycja

$$l.znaczaca * baza^{wykladnik}$$

$$3.82379102 = 382379102 * 10^{-8}$$

Liczby zmiennoprzecinkowe

Liczby rzeczywiste - standard IEEE 754 - binary32

IEEE 754 / binary32

- bit znaku (1 bit)
- wykładnik (8 bitów)
- l. znacząca/mantysa (24 bity, 1 bit nie wprost)
- baza: 2

Liczby zmiennoprzecinkowe

Liczby rzeczywiste - binary32

$$mantysa * 2^{wykladnik}$$

3.82379102

0 (sign)
10000000 (exponent - 1)
11101001011100011111110 (mantissa - 1.9114999771118164)

Liczby zmiennoprzecinkowe

Liczby rzeczywiste - standard IEEE 754 - binary64

IEEE 754 / binary64

- bit znaku (1 bit)
- wykładnik (11 bits)
- l. znacząca/mantysa (53 bity, 1 bit nie wprost)
- baza: 2

Liczby zmiennoprzecinkowe

Liczby rzeczywiste w MIPS

- MIPS posiada 32 rejestry na liczby zmiennoprzecinkowe (32 bity każdy).
- $f0 – $f31
- $f0 nie jest rejestrem specjalnym
- odpowiednie instrukcje korzystają z tych rejestrów dla pojedynczej prezycji
- nie mogą one wykorzystywać rejestrów ogólnego przeznaczenia

Liczby zmiennoprzecinkowe

Podwójna precyzja w MIPS

- wykorzystywanie tego samego zestawu rejestrów, ale parami: $f0 i $f1
- aresowanie poprzez pierwszy rejestr z pary, np. $f0, $f2
- osobne instrukcje arytmetyczne dla różnych rodzajów liczb
  - `add` - l. całkowite
  - `add.s` - pojedyncza precyzja
  - `add.d` - podwójna precyzja

Liczby zmiennoprzecinkowe

Źródła i polecane materiały

- S. Hollasch, IEEE Standard 754 Floating Point Numbers (strona)
- Wikipedia, IEEE floating point (strona)
- H. Schmidt, IEEE-754 Floating Point Converter (strona)
- J. King, IEEE Floating Point Standard (The Implicit 1) (wideo)

Slajd końcowy

Pytania? Komentarze?

Jeśli masz pomysł jak poprawić lub wzbogacić te wykłady,
proszę zgłoś to jako issue w tym repozytorium:

https://github.com/rmhere/lecture-comp-arch-org